April 2013

Waterborne Autonomous Vehicle

Angel Genchev Trifonov
Worcester Polytechnic Institute

Daniel Joseph Miller
Worcester Polytechnic Institute

Edward Charles Osowski
Worcester Polytechnic Institute

ROBOSUB

Waterborne Autonomous VEhicle

A Modular Development Platform for Underwater Robotics at WPI

A Major Qualifying Project Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science
By

Electrical Team	Mechanical Team	Software Team
Ijeoma Ezeonyebuchi	**Sidney Batchelder**	**Daniel Miller**
Breanna McElroy	**Anna Chase**	**Edward Osowski**
Neal Sacks	**Cory Lauer**	**Angel Trifonov**
Adam Vadala-Roth	**Elizabeth Morris**	
	Christopher Overton	
Report Submitted to	Report Submitted to	Report Submitted to
Prof. Susan Jarvis	Prof. Stephen Nestinger	Prof. Michael Ciaraldi
Prof. Craig Putnam	Prof. Kenneth Stafford	Prof. Craig Putnam

4/23/2013

Abstract

This project designed and realized the Waterborne Autonomous VEhicle (WAVE), a submersible modular robotic platform to enable research on underwater technologies at WPI at minimal cost. WAVE's primary design objectives were modularity and expandability while adhering to the regulations for the international competition held by the Association for Underwater Vehicle Systems International. WAVE's core features include a six degree-of-freedom chassis, a modular electronic infrastructure, and an easily configurable software framework.

Authorship

The contents of this report can be classified into four categories: mechanical, electrical, software, and systems level sections. The three sub teams were responsible for writing their respective team sections, while the system-level sections were written by the senior engineers of the term that the specific sections were created in. Additionally, the appendices of this report were created on an individual basis. Although the majority of the sections were created at the team level, each member deserves recognition for their individual work throughout this report. For sections with multiple authors, they are credited alphabetically by last name.

Acknowledgements

WAVE was made possible thanks to the contributions of many outside sources. This section recognizes the people and companies who were critical to this iteration of WAVE.

Sponsors

The development of WAVE would not be possible without the contributions of sponsors. These sponsors provided important donations, either in-kind or monetary. In return, these sponsors were featured in all WAVE signage, and on the team's website: robosub.wpi.edu.

ARM Holdings

ARM Holdings Inc. provided the electrical design team with a one year license of Keil-MDK ARM Micro Controller Development Kit. This software was used for compiling software for the Abstract Hardware Device as well as providing a means to interface with alternate JTAG modules the team received from NXP. Without ARM Holdings Inc.'s generous donation, the embedded software development for WAVE would have been difficult and limited.

NXP Semiconductors

NXP Semiconductors provided all the ARM Cortex M4 microprocessors and JTAG programmers needed to develop the complex embedded computing system aboard WAVE. Each Abstract Hardware Device is powered by a LPC4337JBD144 ARM Cortex M4 donated by ARM, and all embedded software development was carried out using the donated JTAGs. Without NXP Semiconductor's generous donation of microprocessors and programming hardware WAVE's internal electronics would have been far more costly to develop and far more expensive overall. The help of the donations really made the realization of a complete custom electrical system possible.

Vicor Corporation

Vicor Corporation provided three 28V Wide Input Maxi DC-DC converters. These converters supplied an output voltage of 12 volts and an output power of 200 watts. The power system required the Vicor Maxi DC-DC Converters as part of WAVE's 12V conversion board. Without Vicor Corporation's generous donation, WAVE's power system would not be in the current functional state observed in this report.

Gens ace

Gens ace provided three LiPo batteries for the power system. These donations were 10000mAh, 18.5V, 25C, 5S1P LiPo Battery Packs. The LiPo batteries are at the core of WAVE's power distribution system. Gens ace's donation allowed for the proper distribution of power throughout WAVE's system.

Advanced Circuits

Advanced Circuits provided all printed circuit board fabrication required for all of WAVE's custom electronics. In order to realize all of the electrical subsystems required by WAVE, custom printed circuit boards were required for each subsystem. Without Advanced Circuit's donation of their printed circuit board fabrication services WAVE's electrical system would have been far more costly in addition to taking longer to develop, as no other fabrication house is as fast as Advanced Circuits.

IDS Imaging Development Systems

IDS Imaging Development Systems provided two UI-1220LE high performance industrial automation cameras for WAVE's stereo machine vision system. These cameras will be used by future teams to implement object recognition and expanded navigation abilities which will make WAVE a serious contender at AUVSI in the future. Without IDS Imaging Development System's very generous donation of these cameras WAVE would not have a machine vision system and therefore would have poor hopes of success at future AUVSI competitions.

Pololu Robotics and Electronics

Pololu Robotics and Electronics provided a number of generous discounts especially for the team on specific items were needed. Pololu provided discounts on 5 volt DC/DC converters, female pin headers, relay breakouts, and the motor driver for the ballast system. The 5 volt DC/DC converters were used to power the USB hub for connecting AHDs and the cameras. The female headers were used for the AHD's shield connections and pin breakout. The relays and motor driver were used for WAVE's ballast system, the relays being used for solenoid control and the motor driver being used for the positive displacement pumps. Without Pololu Robotics and Electronics' generous discounts the items the team needed would have been far more expensive cutting into the budget for other important parts.

Special Thanks

We would additionally like to thank several individuals in the WPI community for their generous assistance. Kevin Harrington has been extremely helpful throughout the project with helping the team get the Neuron Robotics Software and Bowler Communications Protocol working, and helping with any debugging issues that arose. Greg Overton helped smooth and chamfer the edges of the electronics housing ensuring a good surface to gasket interface. David Ephraim was invaluable to the end-cap machining and CNC-learning process. Alex Camilo has been extremely helpful throughout the project providing the electrical design team with technical expertise in all aspects of electrical engineering and advice in regards to high speed printed circuit board design. Ennio Claretti was very helpful and generous with his time, giving the electrical team access to special purpose soldering tools and assisting with some of the electronic assembly where 1-2 pairs of hands was just not enough. Erik Scott was helpful towards the end of the project working with team members to turn the SolidWorks model into a beautiful 3D render for use in the presentation, the website, and for promoting the project to new students.

Advisors

Lastly we would like to thank our advisors, Professors Michael Ciaraldi, Susan Jarvis, Stephen Nestinger, Craig Putnam, and Kenneth Stafford, without whom the team could not have completed this project. They have all been part of this project since the beginning and their support and investment in the project have been of great help. We are most appreciative of all the time they have contributed to our project while simultaneously advising other projects and teaching courses.

Table of Contents

Table of Figures

Table of Tables

Glossary

80/20®	A framing system using extruded beams of aluminum
Abstract Hardware Device	A general purpose microcontroller with a variety of interfacing options that can be used for a variety of low-level computing and interfacing tasks.
Ant	Apache Ant, a Java scripting library
Basebot	A small 3-wheeled robot used by the software team for testing and debugging.
DC-DC Converter	An electronic circuit which converts a source of direct current (DC) from one voltage level to another
Device file	A file containing the serial number and path of every AHD
DyIO	A hardware USB computer peripheral that allows quick and easy connection between any computer and other hardware peripherals plugged into the DyIO's channels, such as sensors, LEDs, servos, etc.
Electrical Team	The group of students responsible for the design and development of the electrical part of WAVE
fit-PC	The Mini-PC which contains the Ubuntu server, which is used to run the Java code
Future teams	Future WPI MQP group(s) that will take WAVE and build, expand, and improve upon its design
Interface/Poolside Interface/GUI	The interface used to perform different functions on WAVE, such as debugging, communication, running missions, sending commands, etc.
Manager/Task Manager	The function that runs through the XML Mission File and executes the tasks in the specified order asynchronous tasks are started as soon as they are encountered
Mechanical Team	The group of students responsible for the design and development of the mechanical part of WAVE
Mini-PC	A self-contained off the shelf computer; they are the smallest, full-featured computers available, inherently power efficient and contain their own power supply; they are usually not upgradeable
Mission/Mission File/XML Mission file	A collection of synchronous/asynchronous task descriptions written in an XML file to be executed by WAVE
Model	The core of the Model-View-Controller Pattern and represents a particular set of data; WAVE includes several different models such as a Battery Model or a Mission Model
Modular/Modularity	The ability to integrate any individual functional hardware or software into the system
Modules	Components that extend WAVE's functionality in some specific manner designed to be removable and replaceable
PID controller/loop	Control loop feedback controller used in WAVE's control systems
Protocol	An "agreement" between two different modules for how to communicate data.
PWM signal	Pulse-Width Modulation is a modulated signal with variable duty cycle usable, for example, for motor speed control.
Server/Echo server	An echo server used to link the driver library and the LUFA
Software Team	The group of students responsible for the design and development of the

	software part of WAVE
SolidWorks	A 3D mechanical computer-aided design program
Subsystem	The individual electronics that comprise the system as a whole; the ability to add/remove subsystems is what makes the platform modular
Task	The different operations/functions WAVE performs
The Team	If referred from one of the specific group sections it means that specific group, such as ME; if used in a general section, such as Introduction or Background, it means to the whole MQP group

Acronyms

ADC	Analog Digital Converter
AHD	Abstract Hardware Device
AHRS	Attitude and Heading Reference Sensor
AUVSI	Association for Unmanned Vehicle Systems International
AUV	Autonomous Underwater Vehicle
BCP/S	Bowler Communications Protocol/System
CoB	Center of Buoyancy
CoG	Center of Gravity
CPU	Central Processing Unit
CS	Computer Science
DoF	Degree of Freedom
DyIO	Dynamic Input / Output
ECE	Electrical and Computer Engineering
GPIO	General Purpose digital Input/Output
I2C	Inter- Integrated Circuit
IC	Integrated Circuit
IDE	Integrated Development Environment
IDL	Interface Definition Language
IMU	Inertial Measurement Unit
IP	Ingress Protection
ISP	In-System Programming
JAR	Java Archive
LED	Light-Emitting Diode
LUFA	Lightweight USB Framework for AVRs
LiFePo/LFP	Lithium Iron Phosphate/Lithium Ferro-phosphate battery
LiPo	Lithium-Polymer battery
ME	Mechanical Engineering
MIP	Microstrain Inertial Product
MQP	Major Qualifying Project
NRSDK	Neuron Robotics Software Development Kit
NR	Neuron Robotics
NiCd	Nickel-Cadmium battery
NiMH	Nickel-Metal Hydride battery
OS	Operating System
PAFC	Phosphoric Acid Fuel Cell
PCB	Printed Circuit Board
PEFC	Polymer Electrolyte Fuel Cell
PEMFC	Proton Exchange Membrane Fuel Cell
PIC32	Personal Interface Controller 32 bytes
PID	Proportional Integral Derivative
PWM	Pulse Width Modulation
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
RPC	Remote Procedure Call

RTC	Real Time Clock
SCT	State Configurable Timer
RX/TX lines	Receive and Transmit lines
SDK	Software Development Kit
SPI	Serial Peripheral Interface
STAIR	STanford Artificial Intelligence Robot
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
UUV	Unmanned Underwater Vehicle
WAVE	Waterborne Autonomous Vehicle
WPI	Worcester Polytechnic Institute
XML	Extensible Markup Language

1 Introduction

Underwater robots are used in a variety of applications including oil well maintenance [1], deep-sea exploration [2] and underwater ordinance disposal [3]. As such, underwater robotics has become a major field of research and development at many colleges and universities. Most of the collegiate level robotic submersibles are focused on completing challenges as part of a robotics competition but these vehicles also have a number of avenues of potential development as academic research platforms.

Commercial underwater robotic platforms are typically prohibitively expensive for academic research and collegiate Autonomous Underwater Vehicle (AUV) competitions. Furthermore, most commercial AUV platforms are designed for a specific purpose and do not lend themselves to general research. Many of the commercial modular AUVs, such as the Bluefin-12S [4], pose heavy constraints on adding custom modules making them undesirable for research. Remotely Operated Vehicle (ROV) kits such as OpenROV [5] are more economically feasible but lack basic computing and autonomous functionality. These kits typically only have a microprocessor for computing and are not designed to do any complicated calculations or decision-making. Adding on-board computing and custom modules would require additional ad-hoc hardware development to a rigid design.

This project was designed to develop an extensible, open-architecture underwater robotics research platform with the ability to mount custom modules via a universal interface. The Waterborne Autonomous VEhicle (WAVE) is an underwater vehicle capable of autonomous waypoint navigation and obstacle avoidance. The development of a highly reconfigurable and extensible underwater robotics platform eliminates fundamental hardware development, allowing more effective utilization of research time, energy, and funds. WAVE's capabilities beyond basic movement and navigation are defined by the modules attached to it. These modules have standardized hardware and software interfaces that will interface with WAVE electronically. WAVE's design can be partitioned into three primary sections: the

computational core, the chassis, and the modular electrical systems. WAVE is capable of completing a multitude of tasks. This is accomplished through the standardized mounting and electronics system for implementing modules that can be easily integrated with the robotic platform. Thorough documentation on how the system works helps new users understand and modify the system to accomplish the tasks they set for WAVE. This modular design also provides a platform for future projects.

Past Worcester Polytechnic Institute (WPI) Major Qualifying Projects (MQPs) on AUV research platforms shared similar traits to project including economic accessibility and modularity. The IIH1 project [6] emphasized cost effectiveness but suffered a number of defects which hindered usability. Subsequent projects were forced to dedicate resources to fix lingering issues before implementing their original goal. Although this project was conceived separately, it could be considered a successor to the IIH1 AUV. By learning from the errors of the IIH1 project, and by focusing on hardware expandability for future projects, the team can create a more enduring project than the IIH1. This will fill a gap in WPI's research capabilities by providing the university with its first adaptable underwater research platform. This platform will open up several avenues of research that could have far-reaching implications including underwater human-robot interaction and underwater cave exploration.

The remaining sections of this report are organized as follows. Chapter 2 gives a background on AUVs and their components. Chapter 3 identifies and lists the requirements and specifications for WAVE. Chapters 4-6 discuss the design and analysis process of the three subsystems: Mechanical, Electrical, and Software. The integration of these three subsystems is presented in Chapter 7. Chapter 8 discusses subsystem testing and validation. Chapter 9 details future work and Chapter 10 concludes the report. The Appendices provide further details, figures, and code that could not be fit into the report proper.

2 Background

To develop a standardized mounting, electrical and computing system, several criteria pertaining to robotic submersibles need to be evaluated. This chapter will include a brief description of the history of robotic submersibles. It then includes descriptions and analyses of various aspects of their design. These subsystems include the chassis, the propulsion system, the ballast and buoyancy system, thermal regulation, the power system, and the software architecture. Several robotic submarines have been analyzed throughout the chapter, ranging from professionally-constructed to student-made vehicles.

2.1 Robotic Submersibles

Robotic submersibles come in two varieties: ROVs and AUVs. ROVs are Remotely Operated underwater Vehicles that are physically tethered to a vessel while AUVs are non-tethered Autonomous Underwater Vehicles. Both types fall under the term Unmanned Underwater Vehicle (UUV).

Figure 1: An ROV (left, has tether) [7] and AUV (right, no tether) [8]

Most early ROV development was funded by the US Navy beginning in the 1960s to create vehicles for performing deep-sea rescue or to recover objects such as military ordinance on the ocean floor [9]. Following these initial developments, the offshore oil and gas industry began using ROVs to help develop offshore oilfields and by the 1980s were an essential part of the industry as the wells were being drilled at depths too deep for human divers [10]. Since the 1980s, ROV use has expanded to a

number of other fields such as the construction and maintenance of offshore pipelines and other subsea structures, as well as a number of research fields such as searching for shipwrecks [11] and researching marine life [12]. The advancement of artificial intelligence has allowed the robotic submersible industry to expand its development of AUVs in the use of robotic submersibles for tasks that ROVs cannot accomplish [13].

The Association for Unmanned Vehicle Systems International (AUVSI) is the leading international non-profit organization devoted to the unmanned systems and robotics community [14]. They hold a yearly robotic submersible competition whose guidelines are being used in the development of this project. AUVSI was founded in 1972 as the National Association of Remotely Piloted Vehicles (NARPV), triggered by the use of Remotely Piloted Vehicles in the Vietnam War [15]. By the late 70s, the association was called the Association for Unmanned Vehicle Systems, but as the industry expanded through the 80s and 90s, demand for the organization's services became global. In 1996, AUVS officially became AUVSI and now has 2,100 member organizations in 60 countries. The AUVSI Robosub Competition is cosponsored by the US Office of Naval Research and the AUVSI foundation, which is a charitable organization that promotes the educational initiative of the AUVSI. This international competition brings together high school and college students from the US and other countries and offers a monetary prize to the groups that are able to navigate the course and successfully complete objectives. [16]

2.2 Chassis Design Paradigms

AUVs come in a variety of shapes and styles depending on their operating criteria. Many commercial AUVs are torpedo shaped although non-torpedo and even biomimetic chassis are not uncommon. Depending on the shape, an AUV can be designed with different levels of water permeability: pressure vessel style chassis hermetically seal the interior of the craft; flooded shell style chassis allow water to

permeate the interior of the craft; open frame designs have little sense of an interior at all allowing water to flow naturally through the AUV's structure. These different shapes and styles lend themselves to different applications [17].

2.2.1 Chassis Shape

The most prevalent commercial AUV form factor, as shown in Figure 2, harkens to that of a torpedo, a long cylinder. Torpedo designs typically have one rear prop thruster and are steered using control fins. They exhibit low water drag due to a small forward motion profile making them very hydrodynamic and efficient to propel. Due to high efficiency, the torpedo design is the design of choice for missions involving long distance travel where long term drag forces can reduce the range of an AUV with a given energy capacity. Torpedo designs are best suited for open water operation. However, torpedo-designed AUVs have reduced mobility. Although they are capable of 6 degrees-of-freedom (DoF) motion, motion along those DoF are not independent i.e. they are non-holonomic. For instance, to change orientation, a torpedo-shaped AUV must also move forward; their turn radii are rated as a function of their body length. This makes them unsuitable for operation in tight environments or maneuvering around obstacles.

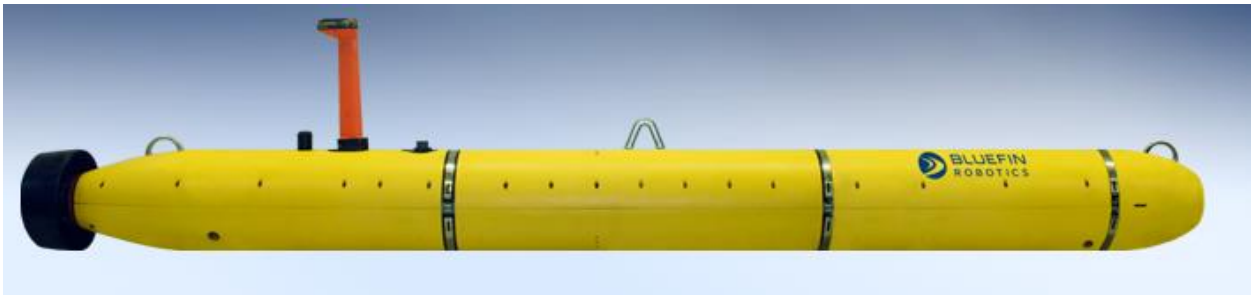


Figure 2: Bluefin-12S - A typical Torpedo Shaped AUV [18]

Non-torpedo shaped AUVs come in all manner of shapes and configurations but generally sacrifice hydrodynamic efficiency for some other desirable property like stability, maneuverability, and modularity. Examples of non-torpedo shaped AUVs are shown in Figure 3. Their less-hydrodynamic

properties make them generally slower and more susceptible to currents than torpedo shaped AUVs, but also more mobile in that they tend to have greater independent control of their DoF. A comparison can be made between underwater and aerial vehicles where torpedo shaped AUVs are to planes as non-torpedo shaped AUVs are to helicopters.

Figure 3: Non-Torpedo Shaped AUVs: SENTRY (left) [19] and ABE (right) [20]

A specific subclass of non-torpedo shaped AUVs are biomimetic designs, which emulate qualities found in the natural world as shown in Figure 4. They typically exploit the high mobility and flexibility of marine animals to create AUVs of similar physical ability. They tend to feature fins, tails, hydrodynamic optimized shapes, flexible bodies and high accelerations. Aqueous biomimetic designs are still a hot research area; as such, there are few such commercial AUVs on the market.

Figure 4: Biomimetic AUVS: DHS's BioSwimmer (upper left) [21], USC's Stingray (upper right) [22], SFIT's Naro-Tartaruga (lower left) [23], and Festo's AquaPenguin (lower right) [24]

2.2.2 Chassis Style

There are three major UUV chassis styles: pressure vessel, flooded shell, and open frame [25]. Examples of each are shown in Figure 5.

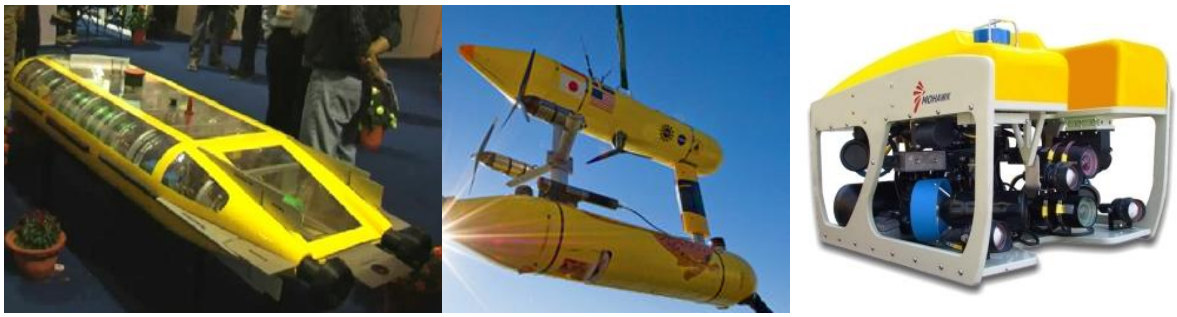


Figure 5: Pressure Vessel (left) [26], Flooded Shell (middle) [27] and Open Frame (right) [28] AUVs.

Pressure vessels hermetically seal the inside of the craft against the external wet environment. These are easy to manufacture and maintain with an easily accessible interior, but are prone to catastrophic failures if the hull is breached or improperly sealed. The space within the vessel is also limited, reducing expandability, and can be prone to internal condensation if not properly handled.

Flooded shells are similar in appearance to pressure vessels. However, they contain hulls designed to let water into the housing. With flooded shells, water-sensitive components are independently waterproofed from the hull. The buoyancy from such a system comes less from the air trapped by the component housings and more from the materials used for or in the hull such as syntactic foam, a machinable lighter-than-water incompressible material. Syntactic foam provides more ruggedness than a pressure vessel while still providing a hydrodynamic shell for water to flow around.

Open frame designs do not protect individual components from the water. They tend to not be hydrodynamic because frames often create eddies and resistance in the water. However, this design is highly expandable, allowing for easy exchange and maintenance of components and modules.

2.3 Propulsion System

Currently, there are three main propulsion systems employed in AUV research and development: propellers, buoyancy driven propulsion, and biomimetic propulsion. [29] [30]

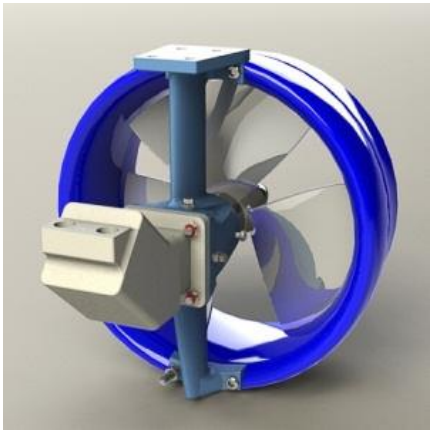


Figure 6: Propeller Based Propulsion [31]

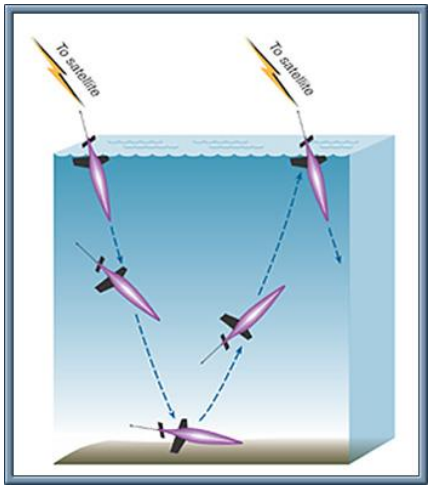


Figure 7: Inertial Propulsion [32] [33]

Figure 8: Biomimetic Propulsion [34]

Propellers provide steady and easy-to-calculate thrust output, but typically lack efficiency. Buoyancy driven propulsion adjusts the mass of the AUV by changing the internal buoyancy of the craft, which allows the AUV to travel up and down in the water while translating along a saw-tooth dive profile. Buoyancy driven propulsion is extremely efficient, allowing for long term missions by affecting slight changes in internal buoyancy of the craft to allow the craft to glide through the water in a saw-tooth path. However, buoyancy driven propulsion systems lack agility in the water because of their restriction to a gliding path, thus they are not used in tight quarter operations. Biomimetic propulsion includes systems that are modeled after living organisms. Biomimetic propulsion is not used in active AUV deployment, but research is being conducted regarding efficiency and effectiveness.

2.4 Ballast/Buoyancy Systems

Ballast designs vary from fluid manipulation control to completely mechanical control. The most common form is the use of a piston to flood and evacuate a compartment with water to change the buoyancy, as used in SeaExplorer [35] and Seaglider [30]. Figure 9 shows a diagram of a piston ballast system.

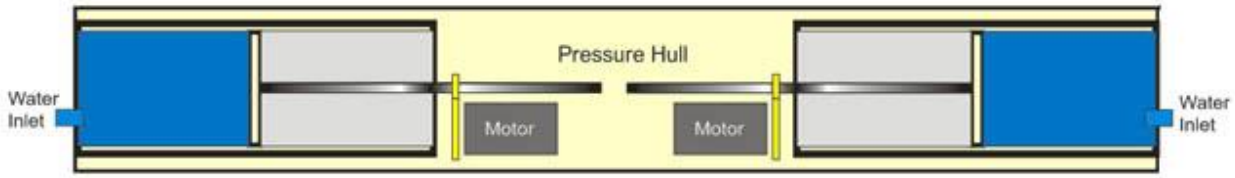


Figure 9: Piston Ballast [36]

Both of these AUV's also use another form of ballast involving the movement of oil into and out of an external bladder. Figure 10 shows a diagram of this bladder manipulation ballast.

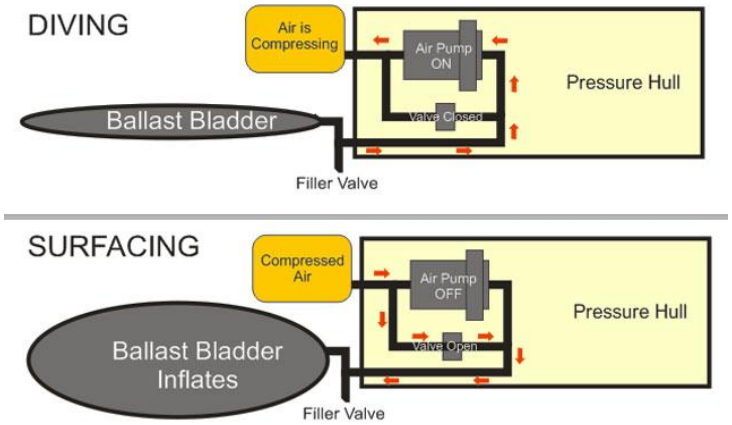


Figure 10: Bladder Manipulation Ballast [36]

While Figure 10 shows the control of air, the same basic principle is applied in SeaExplorer and SeaGlider with the addition of piston control. Oil is forced into the bladder via a piston which increases the volume of the craft, causing it to rise. Some designs use a pump to force water into a ballast tank,

membrane or rigid, to change the buoyant properties of the craft. Figure 11 shows a diagram of a pump ballast system.

Figure 11: Pump Ballast [36]

Others use a limited use system focusing on a container of compressed gas to quickly empty a tank of water. This requires that the gas canister be refilled periodically between missions. Figure 12 shows a diagram of a gas canister ballast system.

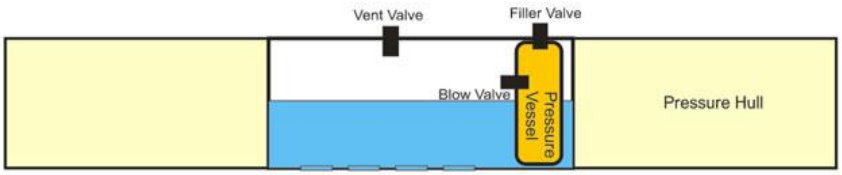


Figure 12: Gas Canister Ballast System [36]

On the mechanical side, the 2010 paper Low Cost Submarine Robot [37] introduced the idea of mechanically expanding the volume of the hull to control buoyancy. A screw motor elongates the hull which is protected by a hydraulic seal, increasing or decreasing the volume as needed. Figure 13 shows an example of mechanically controlled variable volume ballast.

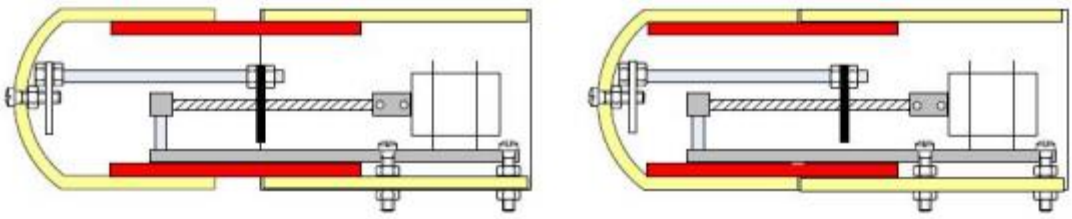


Figure 13: Mechanically Controlled Variable Volume Ballast [37]

There are other systems for controlling ballast that were disregarded due to their need of constant

2.5 Thermal Regulation

Electronic components have maximum operational temperatures, while also emitting energy in the form of heat due to inefficiencies. Convection, conduction, and radiation are the three primary methods of heat transfer. Convection is the heat transfer from an object to its surroundings, due to fluid motion. Conduction is heat transfer through contact. Radiation is heat transfer from the emission of electromagnetic radiation. Through these methods of heat transfer, the vehicle must be able to dissipate enough thermal energy to maintain an acceptable temperature for the electronics to operate safely.

There are different ways to manage heat built up inside the pressure vessels of a submersible. The technology industrial submarines utilize to balance the heat within their vessels is beyond the scope of the project due to size and cost restrictions, but some processes that are used may have application at a smaller scale. One of the main sources of heat generation within a submersible is the power source. Power dissipation from the power source causes overheating which may compromise other subsystems and components in the submarine. For years Bluefin robotics struggled with their battery enclosure. Tests resulted in “dangerous off gassing, the need for refrigeration for maximum cycle, and costly pressure vessels that had to be opened for each charge cycle.” [38] Over time Bluefin settled on using lithium ion batteries in conjunction with an electronic system to control and monitor cell voltages, temperature, and current to create a system less prone to failures and overheating. The system was enclosed in a corrosion resistant, pressure tolerant housing. The challenges faced by Bluefin demonstrate the complexity of maintaining a live battery on a sub, and the importance for appropriate materials and monitors to allow it to run.

2.6 Power System

An important AUV design consideration is the selection of the power source. Many factors play a role in deciding what power source is best for the AUV's application and which characteristics are most important to the AUV including desired speed, type of sensors needed, and length of mission. Multiple types of untethered power sources are used in underwater vehicles including nuclear, combustion, solar, thermal, and electrochemical [39].

2.6.1 Nuclear Power Sources

Nuclear power sources can last for a long period of time without refueling and can operate as a closed system. Though this concept has been proven successful, they are very large, expensive, and highly regulated making them impractical for most AUVs.

2.6.2 Combustion Power Sources

Combustion power sources have also been developed and used for underwater vehicles. While this type of power source is viable for large underwater vehicles, the amount of power and speed attained is not very practical for smaller AUVs.

2.6.3 Solar and Thermal Energy

Both solar and thermal energy have been successfully utilized to support long-endurance missions, such as environmental monitoring and oceanographic surveying [40], with minimal maintenance. This is a more recent advancement in AUV design and is a field for further research and development. Due to these technologies only being in the early stages of development it would not be efficient to use solar or thermal energy in a smaller, inexpensive AUV [41].

2.6.4 Electrochemical Power Sources

Electrochemical power sources, the most common power source for small scale submersibles, use chemical reactions to create electrical energy and can be categorized into four groups: fuel cells, seawater cells, primary batteries, and secondary batteries.

2.6.4.1 Fuel Cells

Fuel cells create electrical energy by the reaction of oxygen and fuel, without combustion. They are larger and more complex than primary and secondary batteries. Examples of fuel cells include phosphoric acid fuel cell (PAFC), polymer electrolyte fuel cell (PEFC), and proton exchange membrane fuel cell (PEMFC).

2.6.4.2 Seawater Cells

Seawater cells utilize the oxygen in the seawater to create a reaction similar to fuel cells. As a result of using its own environment as fuel, it has a high energy density, but consequently has a low power density. It is common for magnesium to be used as the anode material in these cells.

2.6.4.3 Primary Batteries

Primary batteries are portable non-rechargeable voltaic cells that have a high energy density, are safe, easy to use, and are inexpensive. However, they are designed to only be used once and can outgas hydrogen if stored for a long period of time. Examples of primary batteries include alkaline cells and Lithium primary cells.

2.6.4.4 Secondary Batteries

Secondary batteries, portable rechargeable voltaic cells, are the most popular choice for powering AUVs. Secondary batteries have many different trade-off points and should be selected based on the intended application for the AUV. Table 1 compares common types of secondary batteries.

Table 1: Comparison of common secondary batteries [42]

	NiCd	NiMH	Lead Acid	Li-ion	Li-ion polymer	Reusable Alkaline
Gravimetric Energy Density (Wh/kg)	45-80	60-120	30-50	110-160	100-130	80 (initial)
Internal Resistance (includes peripheral circuits) in mΩ	100 to 200[1] 6V pack	200 to 300[1] 6V pack	<100[1] 12V pack	150 to 250[1] 7.2V pack	200 to 300[1] 7.2V pack	200 to 2000[1] 6V pack
Cycle Life (to 80% of initial capacity)	1500[2]	300 to 500[2,3]	200 to 300[2]	500 to 1000[3]	300 to 500	50[3] (to 50%)
Fast Charge Time	1h typical	2-4h	8-16h	2-4h	2-4h	2-3h
Overcharge Tolerance	moderate	low	high	very low	low	moderate
Self-discharge / Month (room temperature)	20%[4]	30%[4]	5%	10%[5]	~10%[5]	0.3%
Cell Voltage (nominal)	1.25V[6]	1.25V[6]	2V	3.6V	3.6V	1.5V
Load Current						
- peak	20C	5C	5C[7]	>2C	>2C	0.5C
- best result	1C	0.5C or lower	0.2C	1C or lower	1C or lower	0.2C or lower
Operating Temperature (discharge only)	-40 to 60°C	-20 to 60°C	-20 to 60°C	-20 to 60°C	0 to 60°C	0 to 65°C
Maintenance Requirement	30 to 60 days	60 to 90 days	3 to 6 months[9]	not req.	not req.	not req.
Typical Battery Cost (US$, reference only)	$50 (7.2V)	$60 (7.2V)	$25 (6V)	$100 (7.2V)	$100 (7.2V)	$5 (9V)
Cost per Cycle (US$)[11]	$0.04	$0.12	$0.10	$0.14	$0.29	$0.10-0.50
Commercial use since	1950	1990	1970	1991	1999	1992

Nickel Cadmium (NiCd) batteries are best for applications where they will be in constant use and actually form crystals on their cell plates if left sitting and not fully discharged. They charge quickly, work in a wide range of temperatures, are inexpensive, and have a high number of charge/discharge cycles. However, they have a low energy density, need to be recharged after storage, are unfriendly to the environment, and are subject to charge memory issues.

Nickel-Metal Hydride (NiMH) batteries have a high energy density, require less maintenance, and are more environmentally friendly than NiCd's, but also have a high self-discharge rate, are less durable, and more expensive.

Lead Acid batteries are inexpensive, reliable, well understood, and require minimal maintenance, but cannot be stored discharged, have low energy density causing them to generally be heavy, have a limited number of charge/discharge cycles, and are also environmentally unfriendly.

Lithium Ion (Li-ion) batteries, when first developed, had many safety issues, specifically during charging. However, research advanced the use lithium ion batteries making them safer and ubiquitous. These batteries still need special care when charging and discharging. Also, they may be subject to aging defects, have transportation regulations, and are costly. However, lithium ion batteries have a high energy density, are low maintenance, have a small self- discharge rate, and are relatively lightweight.

Lithium Polymer (LiPo) batteries are much like their predecessor, Li-ion batteries, but replace the porous separator with a dry, solid polymer electrolyte. Much like a thin film of plastic, the electrolyte allows ions to be exchanged but does not conduct electricity itself. This is the original design of the LiPo battery, but it had poor conductivity and its internal resistance was too high, hindering its effectiveness. To compensate for this, gelled electrolyte is used instead of a solid polymer. Lithium polymer batteries have a high energy density, allowing for thin and lightweight designs. LiPo batteries are low-maintenance, pressure-compensated, and have improved safety but are still more expensive and more dangerous than NiCd, NiMH, and lead acid batteries [43].

2.7 Software Architecture and Framework

The software onboard an autonomous robot is responsible for interpreting sensor data and deciding how it should move or act. These various responsibilities call for a robust computational system, capable

of making time-sensitive calculations and relaying sensor data throughout the system quickly and efficiently.

As there are many different kinds of robots, there are many different software architectures and frameworks used by robots. Some complete robotic software platforms exist to help developers easily create software based on predefined functions through a graphical user interface such as Microsoft Robotics Developer Studio [44]. Microsoft's platform and others, such as Webots, allow for virtual simulation of systems as well. However, there are many robots with requirements that do not fit existing software packages and, therefore, the software must be written from scratch [45].

All robotic systems have some sort of embedded code component for basic functionality. This code is usually written in C or a similar low-level language depending on the electronics board being used. However, these boards do not have the processing power for high-level decision making. Therefore, most advanced robots also include a central processor or computer, running an operating system (OS) such as Linux or Windows, high-level function processing. The on-board computer then runs a high-level program, which will typically use an object-oriented language such as Java or C++. This program handles all of the decision-making and mission control logic. Many AUVSI teams use a similar framework such as Cornell's Nova robot, which uses a custom software stack running on a Linux single-board computer [46]. Some teams also take advantage of the various open-source frameworks for different special functions such as image processing and video capture which are often run on a separate machine from the decision-making machine. A separate processor is used since these tasks are usually computationally expensive and take up too much processor time that could slow down decision making on the primary machine.

One of the key features of any robot software framework is an effective communication protocol between the high-level decision-making machine and the embedded computing system.

Depending on the environment, some teams may implement a custom software framework. However, there are many predefined software frameworks available. One such framework is the Robot Operating System (ROS) [47], developed by Willow Garage. Another framework, the Bowler Communications System (BCS) [48], is quite popular on the WPI campus. The BCS was designed by Neuron Robotics, in tandem with the DyIO, an embedded controller. BCS and ROS both provide strong frameworks for distributed control and computation, essential to any complex robotic system. They are necessary since running all control for the robot on a central processor can result in significant communication delays when sending information to subsystems. By having dedicated module boards control the subsystems; a central processor can focus on the bigger-picture operations for the robot and can send commands and control values to the embedded boards which then do the processing and calculation on their end. The ROS and BCS frameworks have their own strengths and weaknesses, and varying feature sets.

2.7.1 Robot Operating System

ROS is an open-source Robot Operating System developed for the STAIR project by Stanford University [49]. ROS acts as a structured communications layer running on top of a standard Linux environment (typically Ubuntu).It disseminates data through pipelines to its various processes. ROS has five core goals which guided its development: peer-to-peer communication and control, tools-based environment, multi-lingual support, lean implementation, and free availability and open-source [50]. ROS has been natively implemented in C++, Python, Octave, and LISP. Its inter-process communication uses XML-RPC [51]. Very short message descriptions are constructed using the Interface Definition Language (IDL) which is then sent via HTTP. These messages are language-independent; allowing each process to be implemented in whichever language is best suited to the situation.

2.7.2 Bowler Communication System

The BCS was developed by Neuron Robotics [52] and has a similar function to ROS. The BCS was designed to facilitate messaging between Neuron Robotics' DyIO microcontroller and a PC. Like the XML-RPC used by ROS, BCS uses remote procedures calls called Bowler RPCs. A series of procedures are wrapped together into a Namespace, encapsulating a given set of functionality. Each transmission is sent as a series of Bowler packets and pieced together at the receiving end. This layout makes BCS extensible and ideal for communicating between separate microcontrollers.

3 System Overview

Before the individual subsystems were taken into consideration, WAVE was approached from a system's level perspective. This involved creating a concrete set of system-level requirements and specifications along with determining major design decisions that encompass multiple subsystems. Having a well thought-out high-level system alleviates certain challenges from subsystem design. Each individual subsystem would have to adhere to the system-level requirements.

3.1 Functional Requirements of the System

WAVE will serve as a platform for research and collegiate competitions. The functional requirements for WAVE include basic AUV operations such as motion, accessibility, autonomy with sensor feedback, tether control, and safety systems. Further requirements for a completed platform include component modularity for application extendibility and flexibility, wireless communication and control.

3.1.1 Mobility

The mobility requirements including maximum operating depth, mission duration, and maximum speed were determined by looking at what was required to compete in the AUVSI competition and what would be feasible given the budget and material constraints of the project. WAVE must also be able to maintain depth and position using ballasts and motors.

3.1.2 Autonomy

WAVE is required to be autonomous to compete in AUVSI. Full autonomy requires WAVE to sense the environment and navigate to various waypoints. It will need to localize global waypoints relative to its location and determine the best way to navigate to those points as well as perform various tasks. WAVE should also recognize tasks it is unable to accomplish if, for example, certain modules are not connected.

3.1.3 Safety

Safety overrides will be necessary due to the environment WAVE is operating in, as well as the volatility of its batteries. The pressure vessel will require sensors to detect internal flooding, condensation, and critical temperatures. WAVE will automatically stop operations and surface if these sensors detect anything dangerous, as preventing damage to the electronics or the batteries is a top priority during operation.

3.1.4 Modularity

WAVE is a platform intended to enable underwater research and development, as well as serve in various competitions. The system will need to be easily modified in mechanical structure, electrical subsystems, and software infrastructure. WAVE's physical frame should provide an easy interface for adding and removing new components such as manipulators, launchers, sensors, etc. In order to ensure future modularity, the electrical hardware should be able to interface with additional actuators, sensors, and peripherals. WAVE's software could be easily modified to handle different inputs or outputs.

3.2 System Specifications

Table 2 summarizes the desired technical specifications for WAVE, in particular the physical and power constraints.

Table 2: Summary of Operation Specifications (Specs marked with a '†' are from AUVSI guidelines)

Criteria	Value
Maximum Operation Depth	12m
Minimum Run-Time Duration	20 min
Desired Speed along primary axis	0.5m/sec
Maximum Dimensions	0.91m x 0.91m x 1.83m†
Depth Control Accuracy	±12cm
Maximum Mass	54kg†
Maximum Open-Loop Voltage	60V†
Minimum Supply Power	700W

3.2.1 Physical Constraints

WAVE's chassis and pressure vessel must adhere to the AUVSI specifications and fit within 3x3x6 feet (0.91x0.91x1.83 m) and weigh less than 120 pounds (54 kg), out of the water. While AUVSI states its specifications in English units, as an engineering endeavor, WAVE's design specifies metric units.

3.2.2 Mobility

WAVE is intended to be used in calm enclosed bodies of water with a maximum depth of 30 meters. To provide basic motion, WAVE will have at least four DoF comprising a subset of the three translational directions (surge, heave, sway) and the rotational direction of yaw. It will also need to be able to control its depth and position to within ±15.24 meters. WAVE's desired speed was 0.5 meters per second.

3.2.3 Autonomy

WAVE will be able to autonomously navigate between waypoints and perform tasks based on a mission given to it on start-up. Sensor data will be used to negotiate WAVE's surrounding, allowing the vessel to avoid obstacles and collisions, as well as find targets

3.2.4 Safety Overrides

WAVE shall be able to detect any potentially dangerous situations and automatically surface. Users can use an emergency shut off switch to shut down to prevent harm to the robot. Under the event that the emergency shut off is triggered (software, or hardware switch), the sub will immediately cease all motion, and rise to the surface. Also, power from the batteries will be cut off from the system in order to minimize damage in the event of leaking.

3.3 System-Level Design Decisions

While WAVE can be easily divided into mechanical, electrical and software systems, due to the complexity and interconnectivity of these systems, a number of decisions must be coordinated at the system-level to allow successful integration of the subsystems.

3.3.1 Module Design

WAVE was designed as a modular platform to allow for expandability by future teams and to allow it to be used as a research platform with easily modifiable characteristics. By being modular, WAVE has the potential to achieve a multitude of goals and objectives as opposed to being restricted to a small set of requirements and specifications.

To design WAVE, a series of basic requirements for a module were developed. Any module developed for use with WAVE must meet three requirements: the module must be able to easily connect to the chassis frame, compatible with the standardized electronics system, and communicate with WAVE's central processor. For more details on the specifications of modules see section 4.4.

3.3.2 Module Control

The computing and control basis for each module is accomplished using an abstract hardware device, which has been designed to be able to operate as any of a number of personalities based on instructions from the main computer as detailed in section 5.2.2. By having a uniform device, the communications between the modules and the core is easier to control and monitor. All modules of the robot will be controlled by abstract hardware devices. Some will be system critical while others will be less essential. All will be running off identical boards. Modules will be able to be easily added, dropped, or swapped by having a high-level software executive that can easily check which modules are attached and modify its actions accordingly. Future teams will be able to use the standard device functionality to easily design their own modules for use on the robot. All modules that will be used in the AUVSI competition must conform to the size and weight restrictions previously specified, but payloads used simply for research will not have these restrictions.

3.4 System Breakdown

WAVE's architecture can be broken down into three categories: mechanical, electrical and software. The mechanical design encompasses chassis form factor and material, electronics housing, and mobility. The electrical design includes the modular electronics infrastructure, power distribution, and sensing capabilities. The software design covers the distributed processing, mission planning, and human-robot interaction.

4 Mechanical Design and Analysis

The design of the mechanical aspects of WAVE primarily took into account modularity, extensibility, and reliability. Safety, accessibility, and controllability were also considered. Multiple designs and materials were considered for the various aspects of WAVE, primarily including the frame and the housing for the electronics and the external modules. These were analyzed qualitatively through design matrices and quantitatively using calculations, software simulations, and experimentation. The four mechanical subsystems, shown in Figure 14, include the frame, thrusters, ballast, and electronics housing. The approach taken towards the various challenges in the design of the mechanical subsystems is described in this chapter.

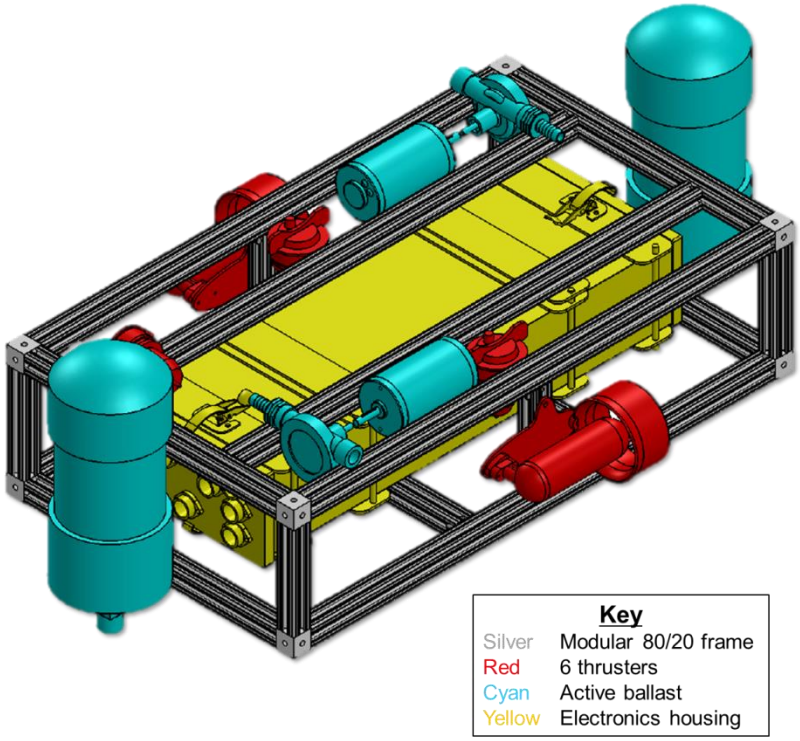


Figure 14: A CAD rendering of the WAVE system depicting the mechanical subsystems.

4.1 System Modeling and Equations of Motion

The hydrodynamics of underwater vessels affect their nimbleness, ballasting properties, and propulsion power requirement. The hydrodynamics forces acting on a submersible vessel are weight, buoyancy, drag, and thrust. A free body diagram of a simple submersed vessel is shown in Figure 15.

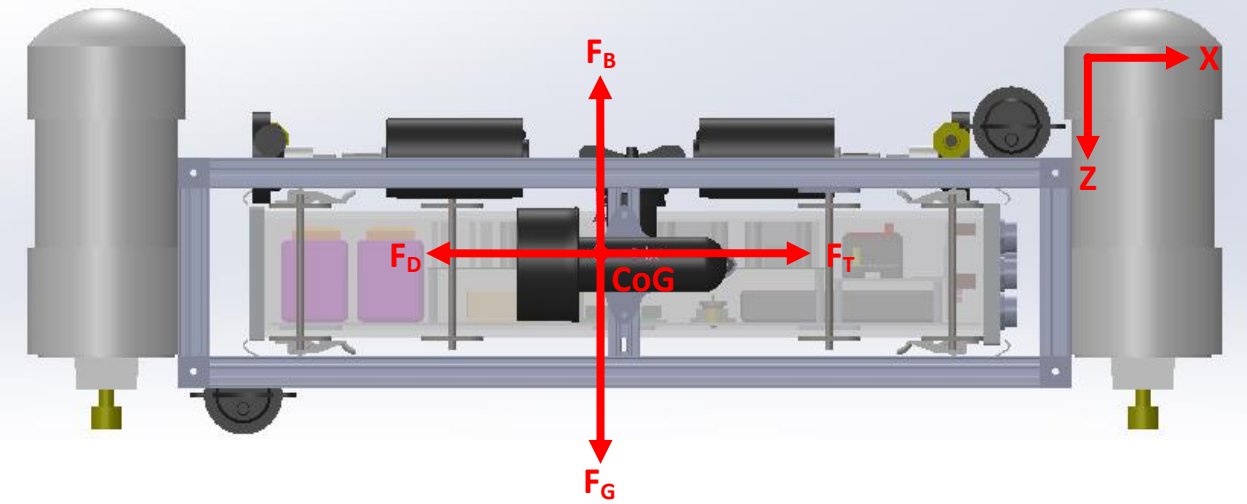


Figure 15: A free body diagram of the hydrodynamics forces acting on a submersible vessel.

$$\overrightarrow{F_g} = -mg\hat{\boldsymbol{k}} \quad \text{(Eq. 1)}$$

$$\overrightarrow{F_b} = \rho g V\, \hat{\boldsymbol{k}} \quad \text{(Eq. 2)}$$

$$\overrightarrow{F_d} = -\frac{1}{2}\rho C_D A_c v^2 \hat{\boldsymbol{v}} \quad \text{(Eq. 3)}$$

$$\overrightarrow{F_t} = m\frac{d^2x}{dt^2}\hat{\boldsymbol{\imath}} + m\frac{d^2y}{dt^2}\hat{\boldsymbol{\jmath}} + m\frac{d^2z}{dt^2}\hat{\boldsymbol{k}} \quad \text{(Eq. 4)}$$

where F_g is the force of gravity on an object, m is the mass, g is the acceleration due to gravity, F_b is the buoyancy force, ρ is the density of water, V is the volume of the object, F_d is the drag force, C_D is the drag force coefficient, A_c is the cross-sectional area of WAVE perpendicular to the motion, and v is the velocity. The thrust, F_t is generated by the propulsion system.

Each of these forces was controlled through design. The weight was determined by the material composition of the submersible, lift by the buoyancy of the craft and its ballast system, drag by the form factor of the craft, and thrust by the available power of the propulsion system.

4.1.1 Weight

The weights of various components included within the craft were known. To determine the overall weight, the component weights were summed. The components include (but are not limited to) the frame, electronics housing, motors, and ballast as shown previously in Figure 14. The weight distribution of the robot can be seen in Figure 16. The total weight is 21.2 kg.

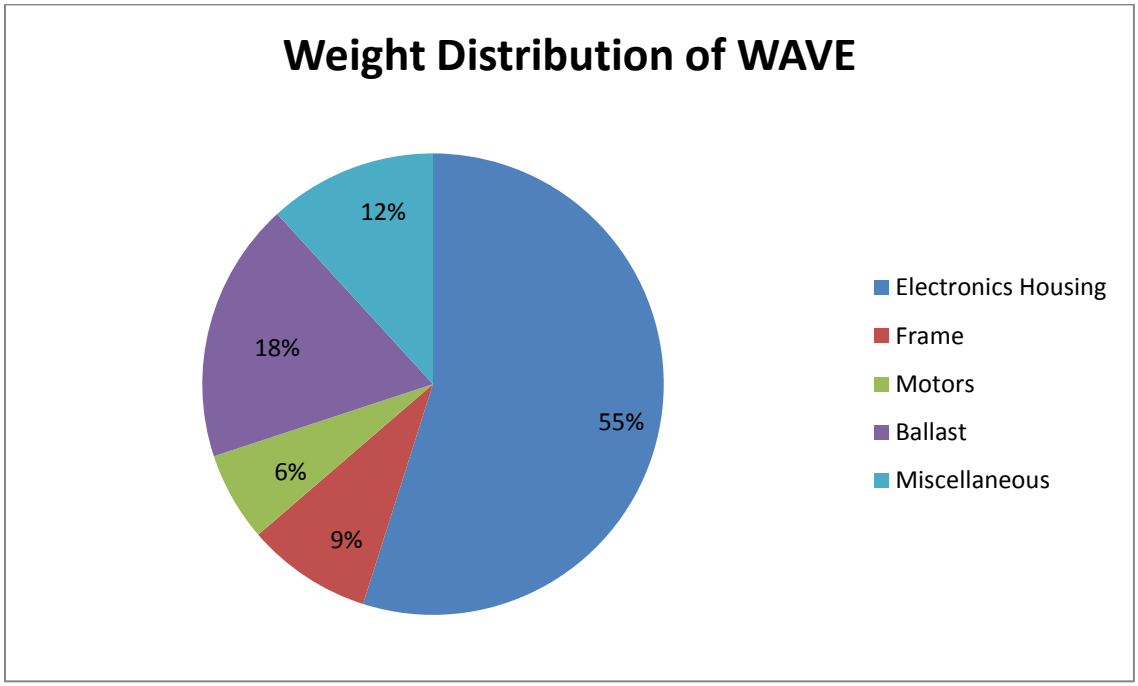


Figure 16: Weight Distribution of WAVE

4.1.2 Buoyancy

The buoyancy of a system is measured by how much the fluid around the system supports the system's weight. For this system, the first step was to find the mass and volume of all components involved. To calculate the buoyancy of WAVE, the mass and volume of each component was estimated using the

values found in the component specifications and in the information posted online by the manufacturers. These calculations occurred during the research phase of the project when the teams were determining which components would best serve the needs of the project. Once these values were determined and summed, the additional buoyancy or weight needed for a particular mission was calculated. These calculations considered the weight and the buoyant force on the system, which were calculated using the equations for buoyant force (Eq. 1) and weight (Eq. 2). This net force equation gives

$$F = F_B - F_g = (\rho g V - mg)\hat{\boldsymbol{k}} \quad \text{(Eq. 5)}$$

From this equation, the volume of material necessary to achieve the desired buoyant force could be calculated. Passive ballast in the form of buoyant form would be placed along the frame to manipulate the CoB as close to the CoG as possible.

The ballast system had two aspects: passive and active. The passive ballast was comprised of buoyant foam and weights strategically placed on the frame. This was used to make the craft slightly positively buoyant before the mission began. The active ballast consists of tanks whose buoyancy could be modified during the course of a mission. It was used for in-mission system balance and buoyancy control. The ballast system was analyzed in order to aid in the placement of buoyancy tanks and weights on the model for the purpose of balancing the craft. Analysis was done manually using buoyancy equations. Although SolidWorks does not have a built-in buoyancy calculation tool, the buoyancy force is given by the weight of the water displaced by the model. The force would be found by changing the material of the parts of the assembly to water, and then using the mass properties function to find the total mass from which the weight could be found.

4.1.3 Drag

Drag force opposes the motion of the vessel and is form-based. The overall design of WAVE is complex. Hence, module placement and vessel shielding will affect the vessel's performance. The drag coefficient of WAVE was first calculated empirically for a simple case, a flat-faced shielded frame using the drag force equation, $F_D = \frac{1}{2}\rho A_C C_D v^2$ with a drag coefficient of 1.05 and a velocity of 0.5 m/s, giving a drag force of 5.4 N. The model used to perform this calculation is shown in Figure 17.

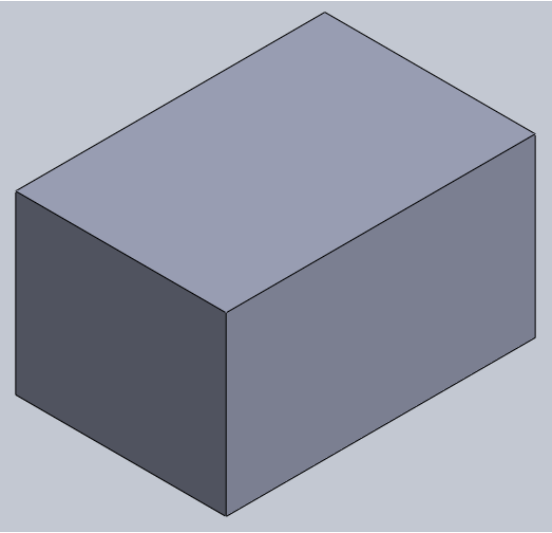


Figure 17: Shielded Frame, Flat-Sided

In order to analyze more complex configurations for WAVE, the Flow Simulation add-in in SolidWorks was used to find the force. An image of a flow figure generated using this software is shown in Figure 18.

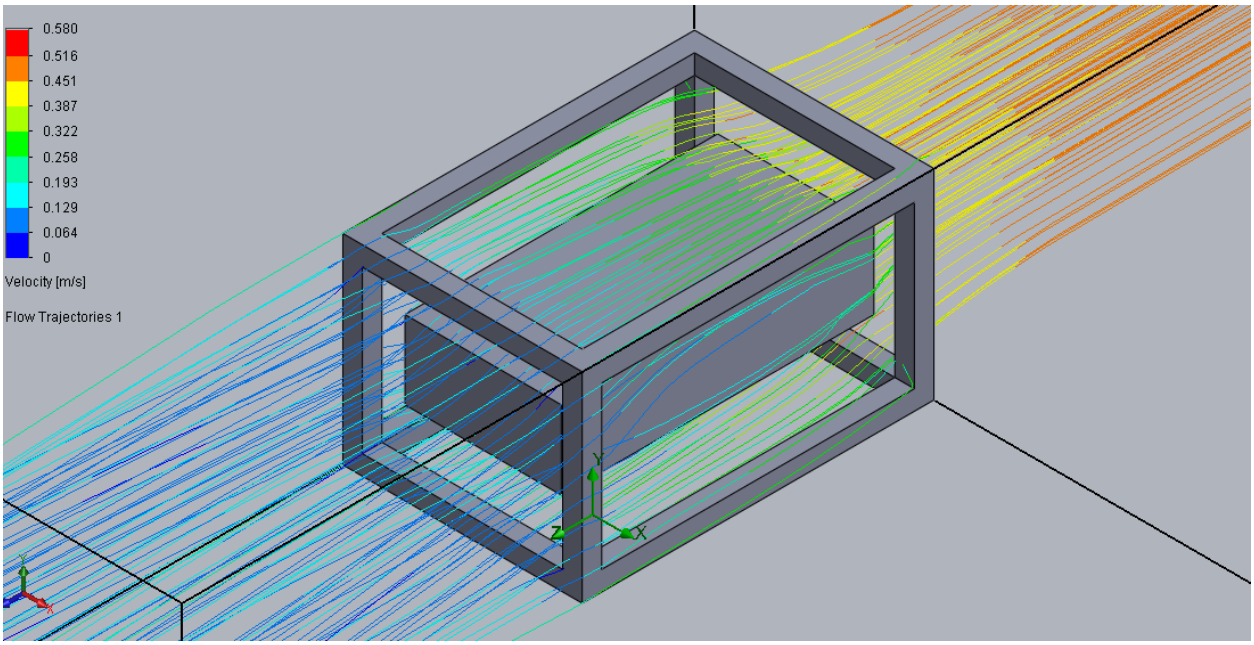


Figure 18: SolidWorks Flow Figure for the Open Frame Model

SolidWorks was used to simulate the same simple-case model. The results from the calculation and the simulation were compared. This comparison allowed for the confirmation of SolidWorks as an accurate way to determine the drag forces for more complex orientations. Once verified, SolidWorks was used to simulate various shapes in order to determine the optimal configuration for the craft. The configurations included various sizes of an open frame model with the electronics housing, a shielded frame with flat sides, and shielded frames in various shapes. An open frame model consists of the frame and the pressure vessel; whereas the shielded models are fully closed off from the water to prevent the water from forming eddies around the various modules. The open frame is shown in Figure 19 while the pyramid and hemisphere frame attachments are shown in Figure 20.

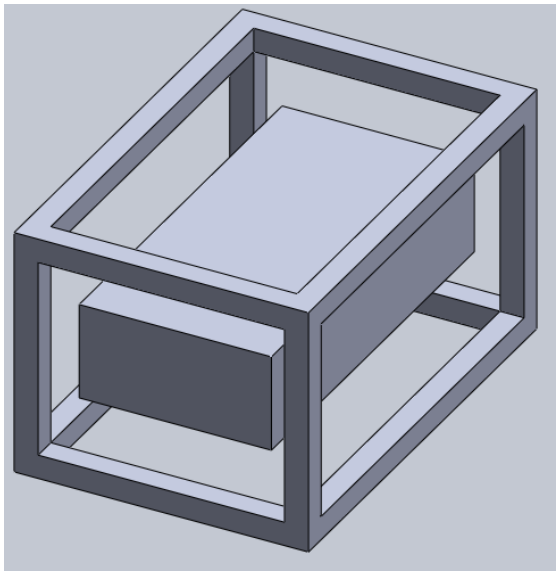


Figure 19: Open Frame Model

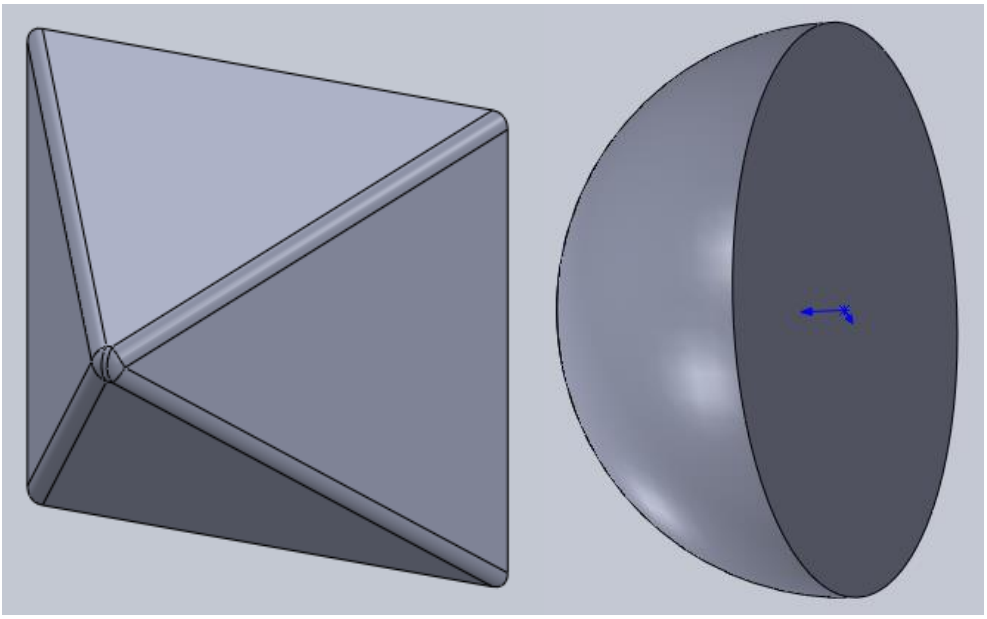


Figure 20: Pyramid Shield (left) Hemisphere Shield (right): attaches to the front or rear of the vessel

Table 3 gives a listing of the various drag force simulations run on simplified CAD models using SolidWorks Flow Simulation software. It gives the type of model, dimensions, cross-sectional area (A_C),

drag force (F_D), drag coefficient (C_D), and drag power (P_D). Not all of the simulations were run at 1 meter per second, because the maximum velocity required by the system is 0.5 meters per second.

Table 3: Drag Force Simulation Results

Type	Frame Size (m)	Electronics Housing Size (m)	A_C (m^2)	F_D (N)	C_D	P_D (W)	F_D (N)	C_D	P_D (W)
				v = 0.5 m/s			v = 1 m/s		
OPEN	0.26 x 0.31 x 0.46	0.10 x 0.20 x 0.41	0.047	7.96	1.38	3.98	31.76	1.38	31.76
SHIELDED Flat-Sided	0.26 x 0.31 x 0.46	N/A	0.081	7.64	0.77	3.82	29.51	0.74	29.51
SHIELDED Pyramid Front	0.26 x 0.31 x 0.46	N/A	0.081	7.09	0.71	3.54	16.35	0.41	16.35
SHIELDED Pyramid Back	0.26 x 0.31 x 0.46	N/A	0.081	4.71	0.47	2.36	18.63	0.47	18.63
SHIELDED Pyramid Front and Back	0.26 x 0.31 x 0.46	N/A	0.081	16.35	1.64	8.18	18.22	0.79	18.22
OPEN	0.38 x 0.38 x 0.61	D=0.10 L=0.51	0.044	4.63	0.85	2.31	--	--	--
OPEN	0.38 x 0.38 x 0.61	D=0.15 L=0.51	0.050	5.92	0.89	2.96	--	--	--
OPEN	0.38 x 0.38 x 0.61	D=0.20 L=0.51	0.069	7.56	0.90	3.78	--	--	--
OPEN	0.38 x 0.38 x 0.61	D=0.25 L=0.51	0.087	7.74	0.726	3.87	--	--	--
OPEN	0.38 x 0.38 x 0.61	D=0.20 L=0.25	0.070	7.25	0.86	3.63	--	--	--
OPEN	0.38 x 0.38 x 0.61	D=0.20 L=0.38	0.070	7.56	0.90	3.78	--	--	--
OPEN	0.38 x 0.38 x 0.61	D=0.20 L=0.51	0.070	8.04	0.95	4.02	--	--	--
OPEN	0.38 x 0.38 x 0.61	0.10 x 0.20 x 0.51	0.060	5.25	0.75	2.62	--	--	--
OPEN	0.38 x 0.38 x 0.61	0.15 x 0.20 x 0.51	0.070	6.67	0.81	3.34	--	--	--
SHIELDED Flat Front	0.38 x 0.38 x 0.61	N/A	0.150	17.35	0.97	8.67	--	--	--
SHIELDED Flat-Sided	0.38 x 0.38 x 0.61	N/A	0.140	12.61	0.71	6.31	--	--	--
SHIELDED Hemispheric Front	D=0.53	0.10 x 0.20 x 0.51	0.230	23.22	0.83	11.61	--	--	--
SHIELDED Pyramid Front and Back	0.26 x 0.31 x 0.46	N/A	0.081	4.117	0.413	2.06	--	--	--
SHIELDED Hemispheric Front and Back, Cylindrical Sides, Flat Back	D= 0.53	N/A	0.228	17.79	0.64	8.90	--	--	--
Type	Frame Size (in)	Electronics Housing Size (in)	A_C (m^2)	F_D (N)	C_D	P_D (W)	F_D (N)	C_D	P_D (W)

After the full system was modeled in SolidWorks, a drag simulation was run on the model. This model included the frame, electronics housing, and the thrusters. The power required to overcome drag force at a constant velocity of 0.5 meters per second was 4.6 watts.

4.2 Chassis

The chassis is a frame that supports all the other aspects of WAVE, including the pressure vessel, thrusters, and modules. The decisions supporting the design and material choices for the chassis are detailed in this section.

4.2.1 Initial Designs

Several different broad design concepts were considered in order to determine the final design. Each of these concepts was non-torpedo-shaped open-frame designs for reasons discussed in Section 2.2.1. Once these concepts were gathered, there was a discussion over the benefits and detractors of each of these preliminary designs. After each design had been thoroughly discussed, an initial design matrix was used to pare down the designs to the three that were viewed as most likely to succeed.

The three models that were considered included a design shaped like an octagonal-puck (Octo-puck), a box-frame design (Boxy), and a rod-attachment based design (Roddy). Octo-puck is shown in Figure 21, Roddy is shown in Figure 22, and Boxy is shown in Figure 23.

Octo-puck's design was thought to increase modularity due to versatile module placement options. This versatility would furthermore allow for the manipulation of the centers of gravity (CoG) and centers of buoyancy (CoB). Wires would have run through its frame.

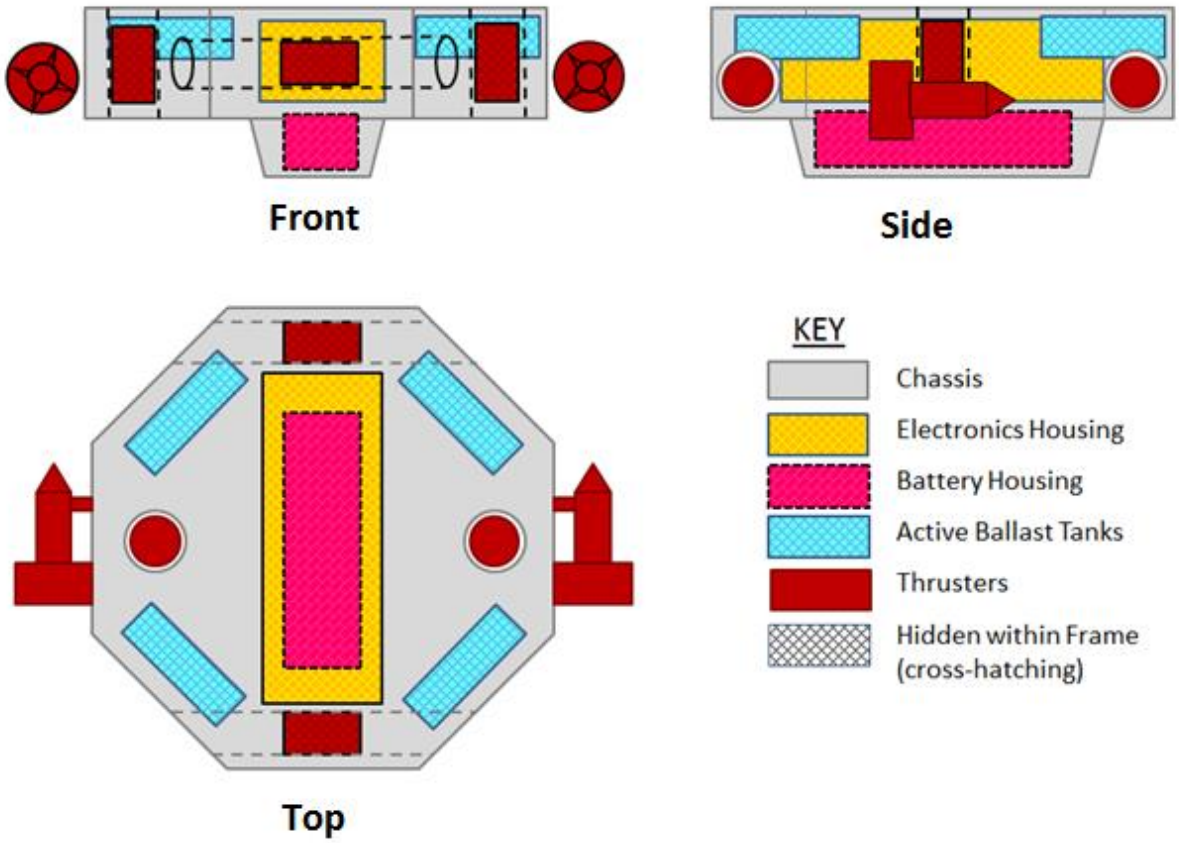


Figure 21: The initial layout for the Octo-puck design.

Roddy consisted of a tube with rods extending from it perpendicularly at each end. The modules connect to the electronics housing by wires run through the rods and would be directly mounted to the two rods.

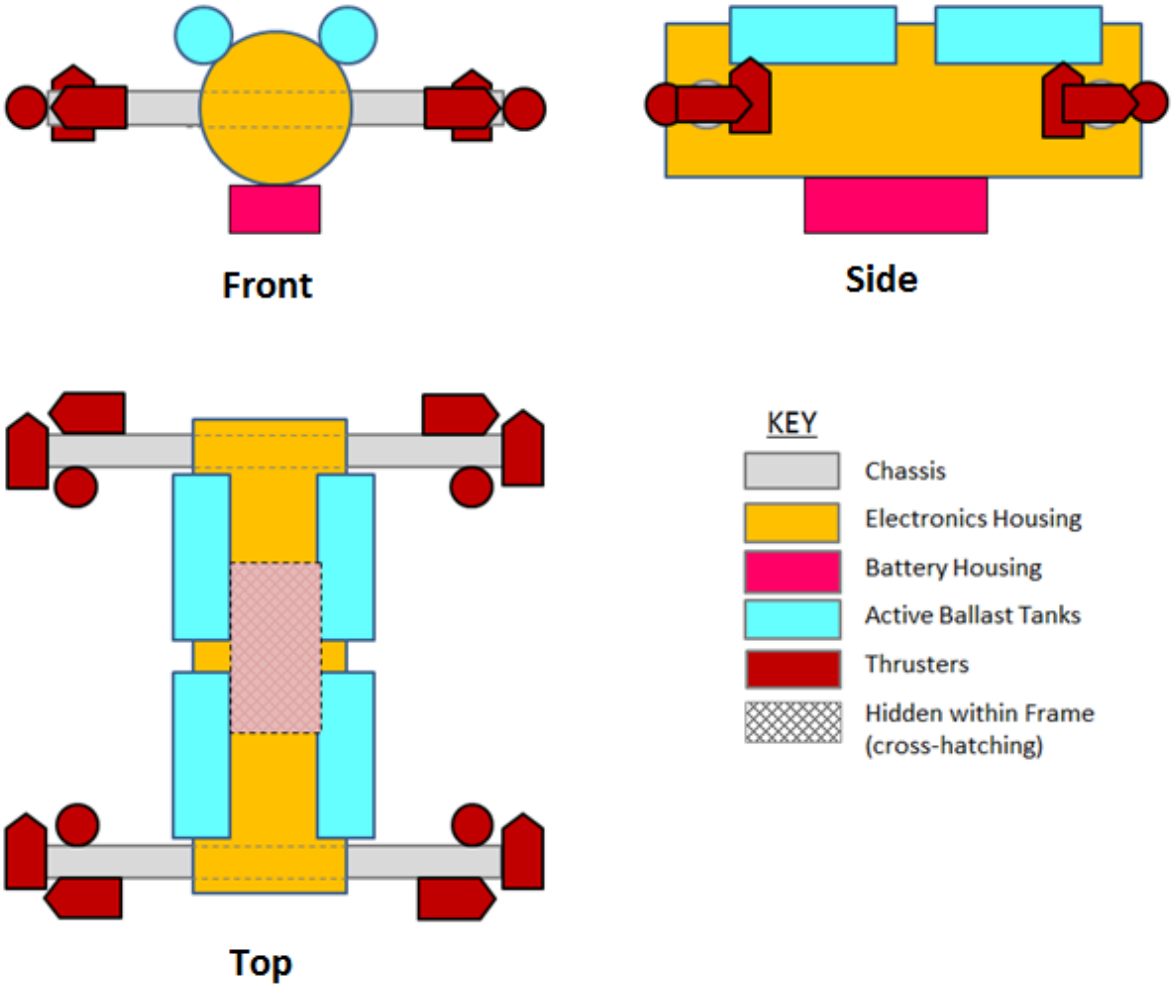


Figure 22: The initial layout for the Roddy Design

The box-frame design had a chassis in the shape of a box with an electronics housing located in the center of the box. Much like Octo-puck, modularity in this model could be accomplished by placing modules at any position or angle within the chassis, which would connect to wires extending from the main electronics housing.

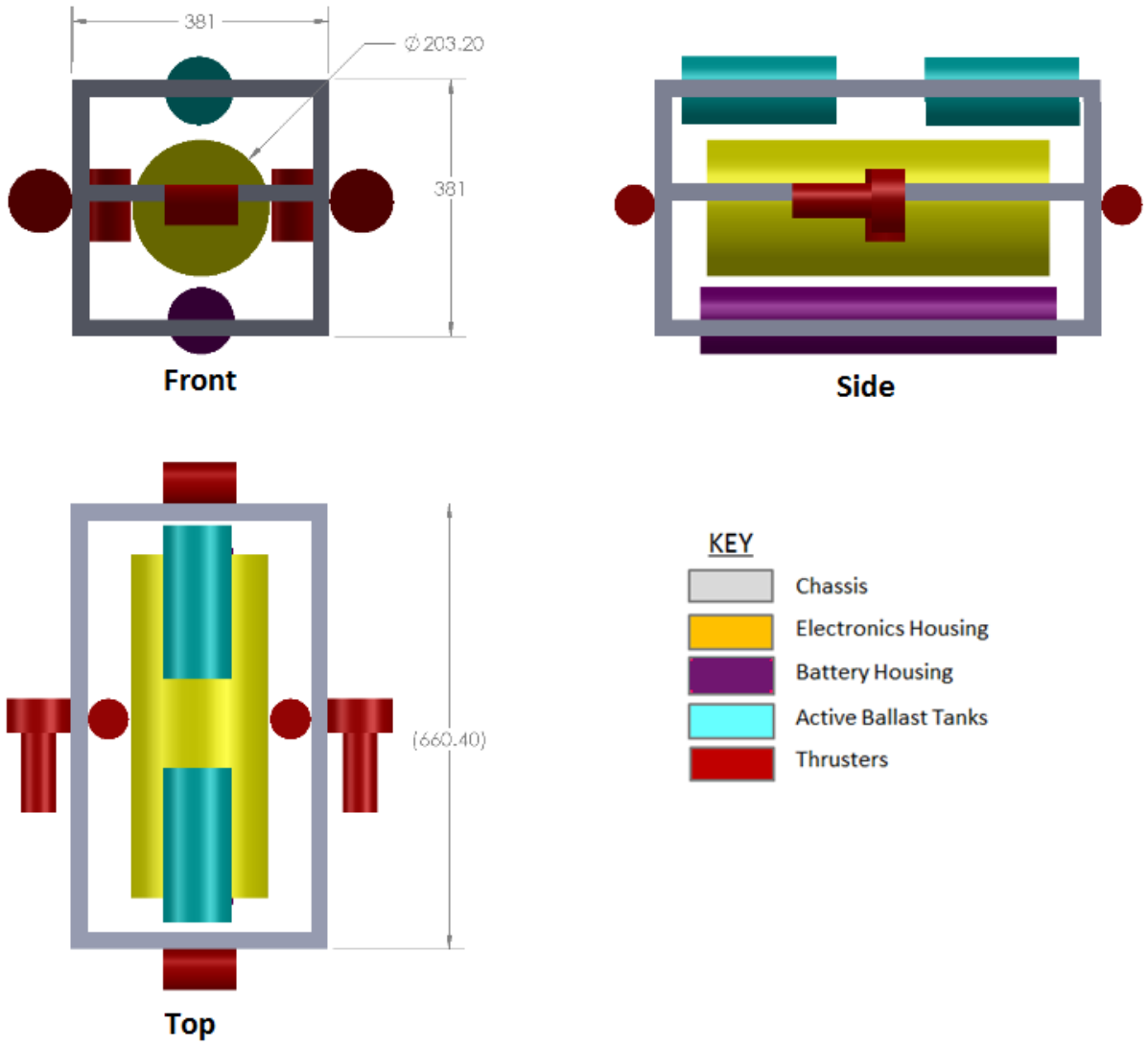


Figure 23: The initial layout for the Boxy design (mm)

A second design matrix was created, which can be seen in

Appendix A: Chassis Design Matrix. The criteria of this decision matrix are seen in Table 4.

Table 4: Design Matrix Criteria

Criterion	Description
Battery Access	How easily the battery can be accessed, removed, replaced, and re-sealed up.
Electronics Housing Access	How easy it would be to get around the chassis to access the electronics.
Module Placement	How easy it would be to add modules to the design.
Part Replacement	How easy it would be to replace parts if they break.
Waterside approach/retrieval	How easy it would be to get the chassis to and into the pool using only members of the MQP.
Ease of manufacturing	How easily manufactured it would be.
Hydrodynamic shrouding	How easy it would be to put a hydrodynamic shroud over the chassis
Hydrodynamics without many modules	How hydrodynamic the design would be without many modules.
Hydrodynamics with many modules	How hydrodynamic the design would be with many modules.
Level of design simplicity	How complex the design would be. This could potentially affect the likelihood of the design to malfunction.
Ruggedness	How resilient the design would be in case of bumping into a wall or a short fall.
Buoyancy Redistribution	How far the buoyancy tubes can be moved around the design.
Rule of Cool	How cool is the design. This is important in that a design that the team thinks is cool and awesome will increase team motivation.

Each of these criteria was weighted, and then rated for each design on a scale from 1 to 10. After this matrix was completed, the design was finalized and further measurements and concepts were started.

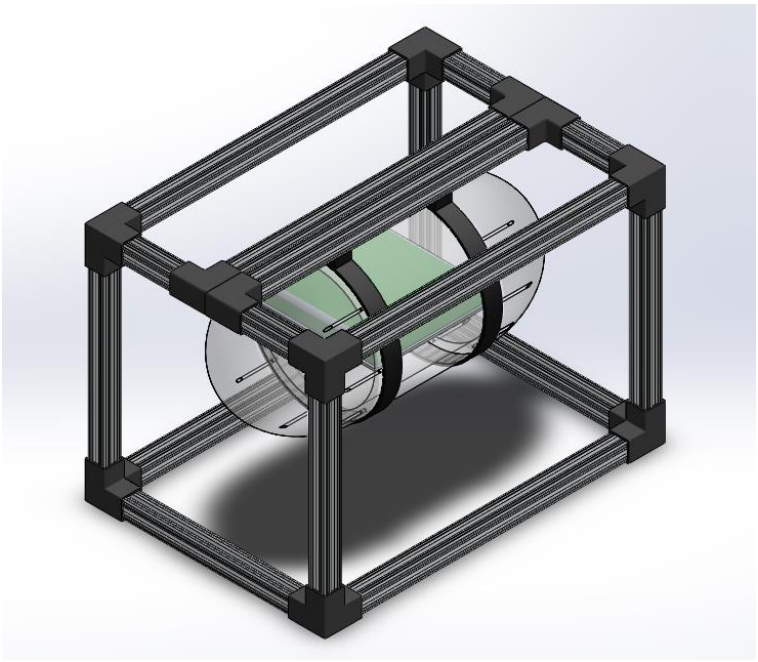


Figure 24: Initial Stage of Boxy's Design

Based on the design matrix outcomes, the Boxy design was selected as the final chassis design. The major contributing factors were manufacturability and inherent modularity.

4.2.2 Materials

Material selection for the frame was based on four major aspects: weight, corrosion resistance, ease of assembly, and ease of using the completed design. The material needed to be lightweight to allow for easier underwater manipulation, corrosion resistant to keep it from degrading over time, easy to assemble and attach modules to, and easy to use. Given these criteria, aluminum was chosen for the frame material because aluminum is corrosion-resistant, lightweight, and readily available. Aluminum is commonly used by AUVSI teams, and is also both cost-effective and easily modifiable. 80/20 Aluminum is known for its modularity, ease of use, and high quality. 80/20 is designed to be simple to put together and attach other components. Parts can be slid on and off of these bars easily through the open slots, as shown in the 80/20 cross-section in Figure 25.

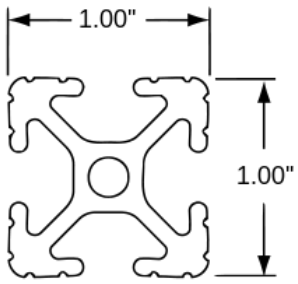


Figure 25: 80/20 Cross Section

Aluminum is a lightweight metal that does not corrode when exposed to water. What little corrosion may occur in chlorinated water can be countered by simply rinsing the metal after testing. Finally, 80/20 is easily accessible and cheap, so those who would use the robotic platform can more easily purchase replacement or custom parts for the simple frame. Additional considerations such as material strength and thermal properties are detailed in later sections.

4.2.3 Form Factor

The frame was constructed to support all of the components of WAVE, including the electronics housing, ballast tanks, thrusters, and sensor housing. The frame is 71.7cm long, 41.3cm wide and 18.4cm high. The layout of the frame is shown in Figure 26.

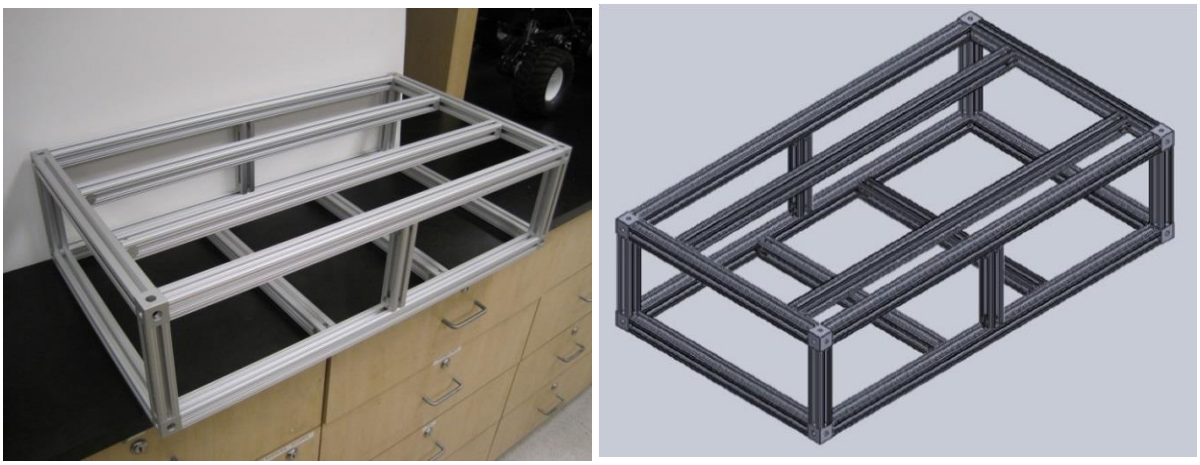


Figure 26: Frame Layout: SolidWorks (left), assembled (right)

There are cross-bars placed along the base and sides of the craft to allow for motor placement. The cross-bars along the top of the craft are placed 20.32 cm(8 inches) apart to allow for mounting the ballast system.

4.3 Electronics Housing

The electrical components of WAVE need to be protected from the underwater environment. This means that many of the components need to be contained within housings. The determination of the number of electronics housings and the physical considerations of the housings are outlined in this chapter. The physical considerations include the shape, size, materials used, the waterproofing, and the thermal aspects.

4.3.1 Waterproofing

Four different methods for waterproofing electrical components were examined: hermetic pressure vessels, bathed pressure vessels, coating using the cast method, and coating using the sprayed method. Pressure vessels surround sensitive elements with a re-sealable casing. Empty space within the pressure vessels can be filled with air, which means the pressure vessel is hermetic, or some other non-conductive fluid like mineral oil, which means that the pressure vessel is bathed. Another method of waterproofing components involves directly coating elements to form a bonded protective non-conductive layer. The types include casting the electronics in acrylic or spraying components with latex. Since electrically insulated materials also tend to be thermal insulators, only low power components tend to be protected in this method. A list of pros and cons for these methods is listed in Table 5.

Table 5: Waterproofing Methods

Method	Pros	Cons
Hermetic Pressure Vessel	Intuitive Available off shelf parts	Must be well sealed to be airtight
Bathed Pressure Vessel	Pressure Tolerant Less prone to water ingress	Messy to access electronics Disliked by Pool Staff

It was determined that hermetic pressure vessels were the best method of waterproofing components because of their ease of component access, organizational capacity, and reduced threat of thermal overload. More information on the design of the hermetic pressure vessel can be found in Section 4.3.3.1.

4.3.2 Form Factor and Materials

The differences between using off the commercial-off-the-shelf (COTS) and self-manufactured pressure-sealed tube for the electronics housing were considered. It was determined that since most COTS waterproof boxes are made of non-thermally conductive plastic, such a solution would put the electronics housing at risk of overheating.

The best shape for withstanding pressure is a sphere, but spherical structures are hard to manufacture or are expensive, especially ones that are hollow and open-able, so this option was disregarded. Two tube shapes were examined for the pressure sealed electronics housing: cylindrical and rectangular.

Cylindrical pressure vessel design is the engineering norm since structurally it is the most appropriate tubing shape for withstanding high pressure differentials because the load of the water is distributed equally around the surface area. It also would not have any places for leaks except at the bottom and top of the cylinder, where the end-caps would be attached.

An analysis was done to see how the loads affected the cylinder, which would determine which material would be best. The stress around the circular perimeter and the stress around the length of the cylinder were found for aluminum, PVC, and acrylic. Aluminum was the strongest of these materials - the stress across the length of the tube equaled $6.68 \cdot 10^6$ Pa, and the stress along the perimeter equaled $3.8 \cdot 10^6$ Pa.

Equations for hoop stress were used to calculate the stress across the length of the tube and the stress around the circumference. First the stress was looked at over 90 degrees. The following formula was used to calculate for this stress.

$$\sigma_{\text{hin}} = \text{P}_{\text{in}} \frac{r}{t} \int_0^{90} \cos(\theta)\, d\theta \quad \text{(Eq. 6)}$$

where P_{in} is the pressure inside the housing, r is the radius, and t is the thickness of the cylinder. This gave a stress of $6.794 * 10^6 Pa$.

To calculate the longitudinal stress the following formula was used:

$$\sigma_{Lin} = P_{in} \frac{r}{2t} \quad \text{(Eq. 7)}$$

This gave a stress of $3.8 * 10^6 Pa$.

Given these values the von Mises stresses were calculated, then plugged into the Distortion Energy Theorem to calculate for the factor of safety. The following formula was used to calculate for the von misses stress.

$$\sigma_v = \sqrt{\sigma_1^2 + \sigma_3^2 - \sigma_1 \sigma_3} \quad \text{(Eq. 8)}$$

Setting the hoop stress along the circumference as the first principle stress, σ_1, and the longitudinal stress σ_3, the von misses stress, σ_v, was found to be $1.157 * 10^7 Pa$. The von misses stress is then plugged into the distortion energy theorem in order to find the safety factor.

$$SF = \frac{S_{y,Al}}{\sigma_v} \quad \text{(Eq. 9)}$$

This formula takes the yield strength of aluminum, S_{yAl}, into account while solving for the safety factor. When yielding occurs in any material, the distortion strain energy per unit volume at the point of

failure equals or exceeds the distortion strain energy per unit volume when yielding occurs in the tension test specimen. This formula was important to explore in this situation because water can cause buckling in this case. For the aluminum cylinder, the factor of safety equaled 23.8. This is in comparison to acrylic which equaled 21.236, and PVC which was 20.629. Though PVC was a cheaper solution, and acrylic would allow the electronic parts to be visible, aluminum was chosen because of modularity and extensibility. 80/20 brand aluminum is easier to mount future modules to in the future.

A rectangular tube shape was chosen because it can be used more efficiently for electronic components because these components are rectangular. Due to the inefficiencies caused by the shape, the cylindrical pressure vessel would need to have a significantly larger volume than the rectangular tubing, and would therefore generate substantially more buoyant force based on (Eq. 2). Although the rectangular tubing is theoretically more prone to failure at high pressures, it was assumed then tested that a rectangular tubing of sufficient thickness would be able to withstand pressures at the maximum pool depth because the factor of safety was so high for the cylindrical tube in the hoop stress calculations.

The size of the electronics housing was driven by the single largest housed electronic component (the fit-PC) which measures 16cm X 16cm X2.5cm. This measurement gave the minimum allowable width of the housing face. The length and height of the design was determined in SolidWorks by arranging the other internal components of the housing and creating the shape such that it encompasses all the components.

4.3.3 Thermal Considerations

The electronics enclosed in pressure vessels release heat as a byproduct of their operation. In an enclosed space, heat from electrical components can build up beyond the tolerances of the components. This is especially true of high current components such as motor drivers. With the goal of

keeping the design of the housings simple, these components are made out of thermally conductive material to promote natural heat exchange with the exterior of the craft. This ruled out materials such as acrylics or plastics. Since the material of the craft must also be rigid and resistant to harmful corrosion, the material selected for the housing is aluminum which exhibits the desired properties and is economically available.

General equations for heat transfer were used to predict the ability for the aluminum housing to dissipate thermal energy to the surrounding water. Shown below are the equations used

$$\frac{1}{U} = \frac{1}{h_1} + \frac{d_w}{k} + \frac{1}{h_2} \quad \text{(Eq. 10)}$$

$$q = UA\Delta T \quad \text{(Eq. 11)}$$

In (Eq. 10) and (Eq. 11), U is the heat coefficient of the system, h_1 and h_2 are the convection coefficients of the fluids on either side of the aluminum (air and water), k is the thermal conductivity of aluminum, d_w is the thickness of the aluminum, A is the surface area of the aluminum, ΔT is the difference in temperature of the fluids, q is the energy dissipated by the system [53]. After plugging in values, assuming the inside of the electronics housing is 70°C and the water is 27.8°C, approximately 298W of energy would be dissipated through the housing. The main electronics components generate roughly 300W of power, which means that around 70°C the system would be dissipating as much energy as the electronics are generating.

4.3.3.1 Thermal Simulation

In addition to testing, thermal analysis on the electronics housing was conducted using ANSYS. A simplified model of the components, as well as blocks representing air and water, was imported from SolidWorks. Materials and worst-case heat generation values were assigned to the various components. The final simulation results are shown in Figure 27.

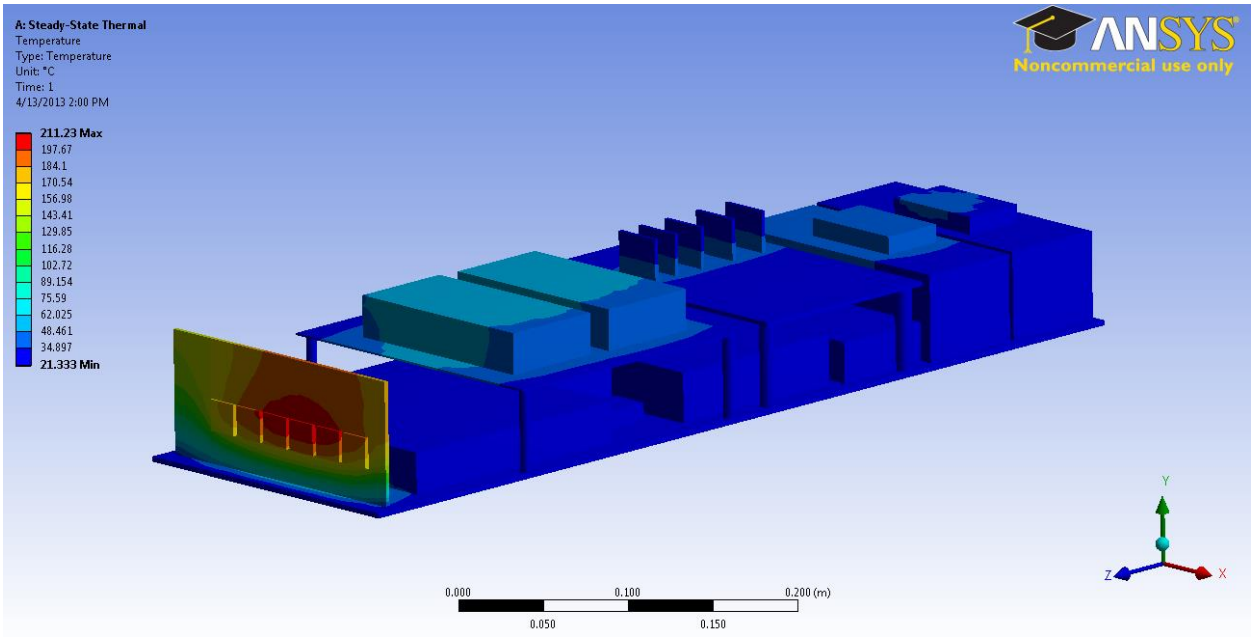


Figure 27: ANSYS Simulation Results

The temperatures within the housing range from 21°C to 211°C. This number was achieved using a Steady-State Thermal Analysis, which doesn't depict time elapsed. Thermal testing will negate these numbers, as explained below.

4.3.3.2 Thermal Testing

Thermal testing of the pressure vessel was performed to determine the temperatures at various points in the housing using convection and radiation as the primary sources of heat transfer. For this experiment, it was assumed that conduction between the components and the housing would be negligible. These tests were performed in air, rather than in water. A resistive heating element with a resistance of 27.6Ω, a thermistor, and a thermocouple were used for this experiment. The thermocouple was used to verify the values received from the thermistor circuit. The element was powered through a wall outlet at 120V. The element had a dial that would adjust the percentage of voltage that was allowed through the system. This percentage could be calculated by finding the voltage required to generate power. This is done using Ohm's Law, $P = \frac{V^2}{R}$. For the set-up of the experiment, a resistive

heating element was placed into the housing on ceramic tiles to prevent heat transfer from conduction where the element contacted the housing. The thermistor was wired into a voltage divider with a resistor of 120Ω, and was powered using a National Instruments DAQ Box (NI USB-6229). This circuit can be seen in Figure 28.

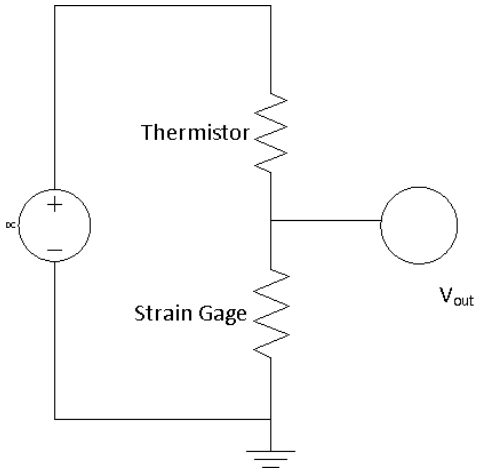


Figure 28: Voltage Divider Circuit

The thermistor was then placed against a wall of the housing as shown in Figure 29.

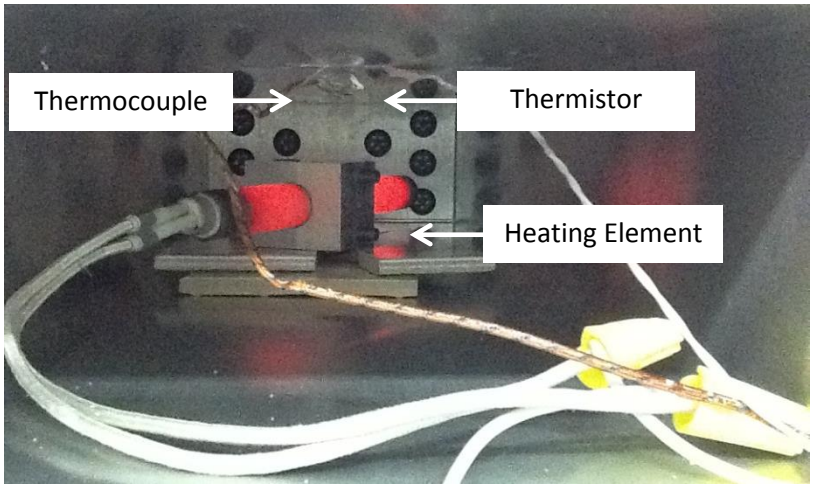


Figure 29: Heating element with thermistor and thermocouple touching the surface of housing.

The first round of testing generated 500W of heat in the center of the tube using the heating element. This simulated the worst-case heat generation by the electronic components. The various placements of the heating element and the thermistor are shown in Figure 30 and Figure 31.

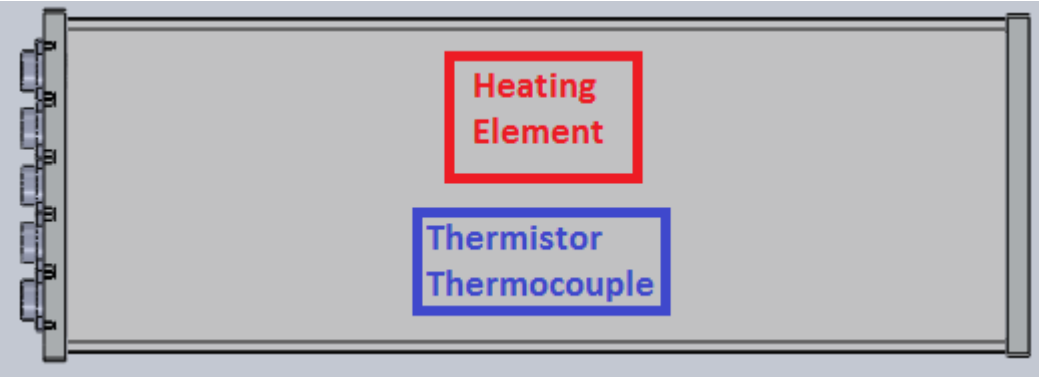


Figure 30: Thermal Testing Set-Up, Run 1

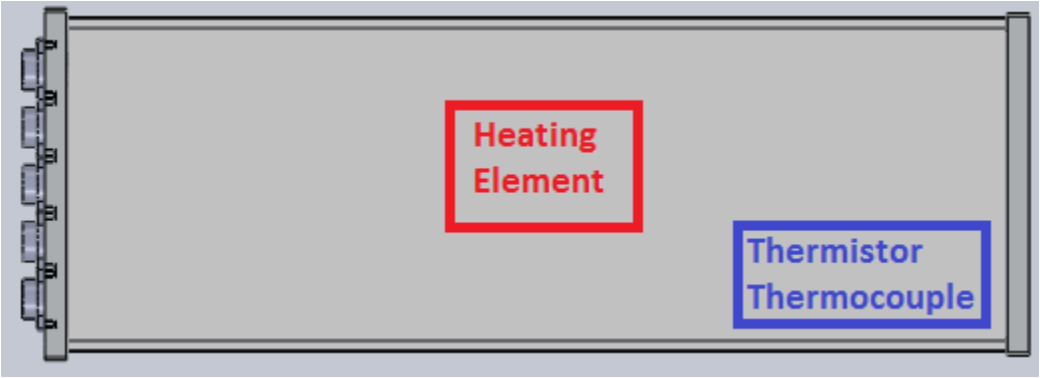


Figure 31: Thermal Testing Set-Up, Run 2

In the first run, the thermistor was placed towards the middle of the tube; in the second run, the thermistor was placed off-center, nearer to an end-cap; and in the third run the thermistor was placed next to an end-cap . The temperature increased over time, until it reached steady state. This can be seen in Figure 32, which shows a plot of the temperature versus time for the second placement of the thermistor.

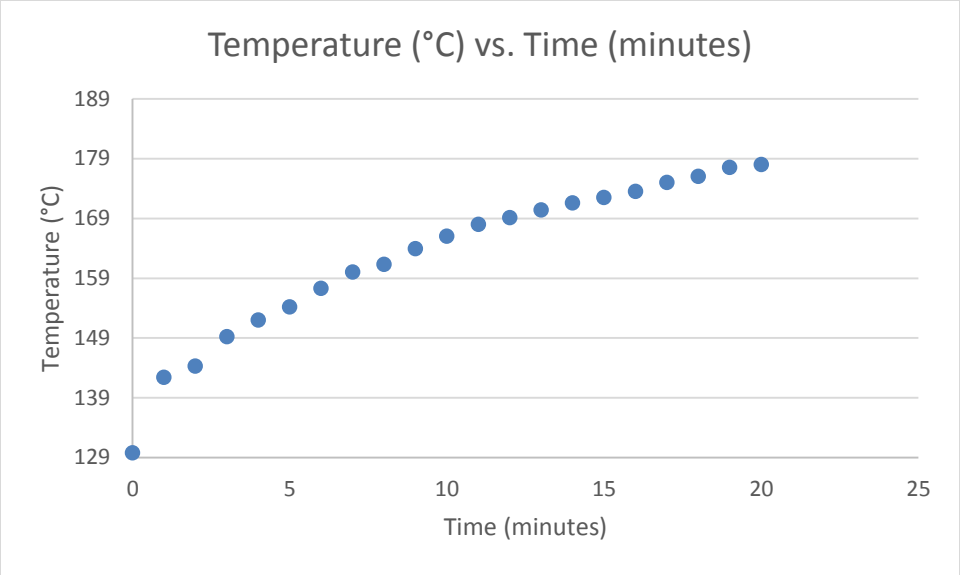


Figure 32: Temperature vs. Time Plot

The steady state temperature was 185°C right by the heating element and 178°C at the end of the housing.

The next round of testing was used to simulate the effect of the motor driver board, which is the largest contributor to heat generation, on the batteries, which are the most volatile, heat-sensitive component. The set-up for this test is shown in Figure 33.

Figure 33: Set-Up of Second Round of Testing

Worst-case, the motor board released 300W of power. The board is located right next to the end-cap with connectors in it to allow for wires to go through easily, and the batteries are located on

the opposite side of the housing. The resistive heating element was set up to generate 300W of heat at one end of the housing, and the temperature was measured at the opposite end of the housing. The final steady-state temperature was 182°C.

This test was performed in the open air at approximately 22°C. Air is significantly less thermally conductive than water. In water, the heat would dissipate from the housing to the outside water much more efficiently, thus keeping the internal temperature lower.

4.3.4 End-caps

The end-caps were a critical design component of the electronics housing. Like the housing-tube, they were made out of aluminum which provided thermal conductivity, resistance to corrosions, and a rigid material to mount electrical through-connectors to. Other materials, specifically acrylic, was considered but was rejected due to its brittleness which could cause cracks that would create leaks. Other plastics were briefly considered but were incapable of matching the thermal properties of aluminum. Other metals were not considered to avoid potential difficulties of using mixed metals in thermally variant, salinized bathed environment.

The end-caps were designed as plates capable of fitting over the ends of the aluminum tubing. They feature a groove that allows housing walls-ends to snuggly fit in it, effectively 'capping' the tube-end. The plates were 13mm (½ inch) thick to ensure rigidity despite multiple connector through-holes, to provide enough material was available for bottom-tapping screw holes to secure connectors, and to accommodate the 1 cm deep machined groove. The end-caps were manufactured in the WPI Washburn shops using CNC mills based on CAD models. The corners of the groove were kept rounded so as to avoid stress concentrations on the silicone seal, described next.

The grooves were partially filled with Sylgard-184, a two-part curing silicone elastomer soft enough to compress under the attachment load between the end-cap and the housing tubing. It was

experimentally determined in section 8.1.1.1 that a half-filled groove (5mm) provided both enough silicone to hermetically compress and enough groove wall-surface to secure the ends of the aluminum tubing. To calculate the amount of silicone to use, the cavity-volume of the groove was determined using, and then halved, as seen in (Eq. 12) and (Eq. 13).

$$V_{groove} = D(2L_1W + 2L_2W + 4W^2) \quad \text{(Eq. 12)}$$

$$V_{silicone} = \frac{1}{2} V_{groove} \quad \text{(Eq. 13)}$$

where D is the depth of the groove (10mm), W is the width of the groove (8.4mm), and L_1 and L_2 are the inside lengths of the grooves, as shown in Figure 34 below.

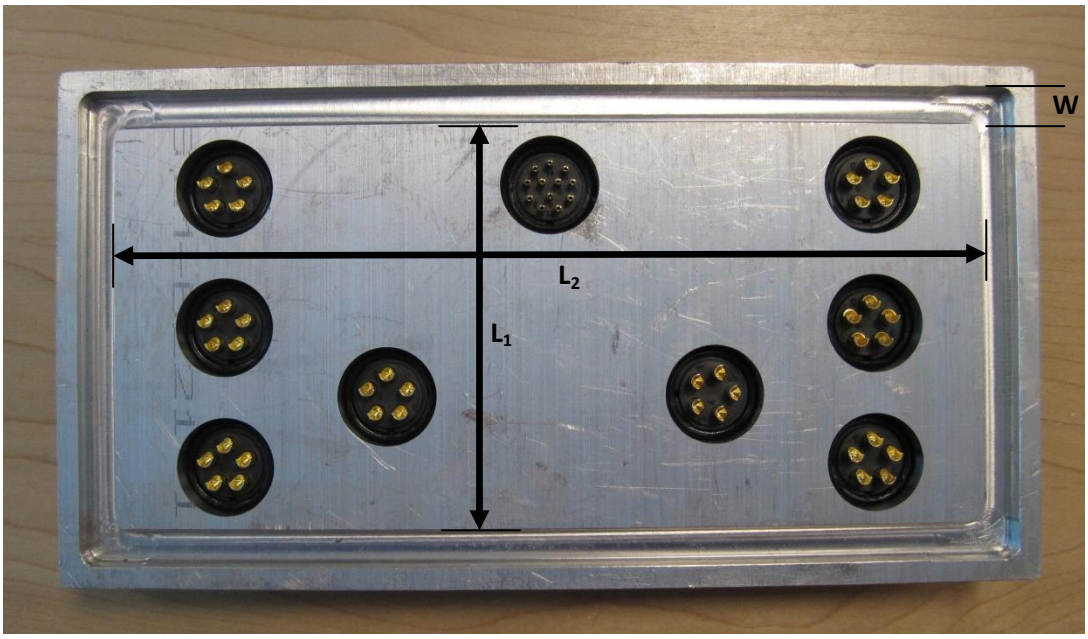


Figure 34: End-cap, labeled dimensions

The unit of volume used was cm^3 which was is easily converted to liquid volume unit ml used to measure out the silicone curing reagents. Sylgard-184's two-part curing elastomer specifically needs to be mixed in a 1:10 ratio so the total needed volume was divided by 11. One part of 11, the curing agent

(typically about 1.5 ml), was measured using a syringe and the 10 parts base silicone gel was measured out in a small graduated glass. The compounds were mixed, poured evenly into the groove, and allowed to cure on a horizontal surface for 2 days.

The in-house made silicone gasket was used over O-rings because, although O-rings are a tried and true sealing medium, the rectangular nature of the end-caps and housing did not have provided optimal geometry for O-ring use.

Two methods of securing the end-caps to the electronics housing were explored: ratcheting straps and latches. Ratcheting straps were beneficial during testing due to their simplicity and effectiveness however the desire for individually removable end-caps obviated their use. Latches that hook the end-caps onto the frame were ultimately used in the design because they allowed for individually removable end-caps and were perceived to be less cumbersome to use regularly.

4.3.5 Connectors

One of the end-caps was designed to accommodate connectors that would allow for an electrical interface between in inside and outside of the electronics housing. To prevent leaks the connectors had to be waterproof. This lead to the selection of IP68 rated connectors of various types (see Appendix L: for definition). IP68 USB and Ethernet connectors were used, as well as WEIPU pin connectors. 5pin and 12pin connectors were selected for variety and expandability. The 5pin connectors are rated at 30amps per pin allowing for high current power transmission for thrusters and the like, while the 12pin connectors allowed multiple channels for signal transmission. The pin connectors are shown in Figure 35. The layout of connectors on the end-cap was modeled to ensure that enough space exists on the face of an end-cap to accommodate the needed number of connectors.

Figure 35: 5 and 12 Pin WEIPU Connectors.

4.4 Modules

WAVE is designed based on modularity, meaning that auxiliary components may be added to provide functionality for specific mission requirements. The use of 80/20 allows for components to be mounted and dismounted with ease. Provided that modules are inherently neutrally buoyant, they can be placed on any spot on the frame without having to modify the existing buoyancy system. The system for attaching these components to the chassis in terms of distribution and housing is described in the following sections.

4.4.1 Distribution

A driving criterion for determining the housing system for these components was distribution. In a distributed system, each component would be kept in its own personal, waterproofed enclosure. In a centralized system, all of the electronics would be kept in a single pressure vessel.

A distributed system would allow components to be placed anywhere within WAVE's frame, allowing for many mass distribution options which could be used to create different stable sub configurations. Having many pressure vessels for components increases the total number of pressure

seals, which increases the risk of leaking. If one vessel was to leak and suffer a catastrophic failure, only one component would be damaged. Another consideration is that a distributed system would expose more wires to water and would use many more connectors, increasing the cost and complication of the craft.

A centralized system features all of the electronics in a single pressure vessel. This limits the range of locations on the submersible where the housing could be fastened to ensure a balanced craft. Additionally, if the pressure vessel were to suffer seal failure, the entire electronics contents of WAVE would be jeopardized. A seal leak would be less likely to occur than in a distributed system because it would only have a single seal point of failure rather than many.

A hybrid distribution system, in which most of the electronics were kept in a centralized pressure vessel except for external sensors that are sensitive to electrical noise, was determined to be preferable. The decision to keep certain external electronics in their own pressure vessels was chosen because some sensor signals can be sensitive to electrical noise, especially over longer cords. An external hub for peripherals allows for signal processing before the signal attenuates. Overall, such a setup results in a safe and effective system.

The determination of the battery housing was a difficult decision. The battery could either be kept in its own housing or within the primary pressure vessel. Keeping the battery in a separate container allows for weight configurability, replaceability, and safety. Furthermore, an external battery would allow WAVE's battery to be swapped out or charged without opening the electronics pressure vessel and exposing its insides to a potentially humid exterior. Since lithium polymer batteries can be volatile, a battery contained within its own housing would not physically damage the other electronics in the event of failure. Alternatively, housing the battery within the electrical pressure vessel would mean fewer connectors through housings and fewer wires exposed to water. The costs relieved by the

reduced number of connectors and containers are substantial. Issues with excessive exposure to humidity could be combatted by using desiccant packets or another absorptive material to contain the excess moisture. For these reasons, the battery is not contained within its own housing.

Neutrally buoyant modules had a negligible effect on the stability of the system. Modules which were not neutrally buoyant, such as Due to the attempted neutral buoyancy requirement imposed on modules, their placement and distribution on the frame was trivial from a stability point of view. Modules which don't comply with the thrusters and batteries were placed on the frame so as to increase craft stability: heavier-than-water parts are prioritized to be low in the frame whereas lighter-than-water components are held higher. In the case where placement was critical, passive ballast, including buoyant foam and weights, were added to the frame to compensate for craft imbalance.

4.5 Thrusters

Thrusters were needed to propel WAVE through the water. They were selected based on the estimated power required to move WAVE at the specified 0.5 m/s velocity considering the drag forces on the craft, as was shown in section 0 about The ballast system had two aspects: passive and active. The passive ballast was comprised of buoyant foam and weights strategically placed on the frame. This was used to make the craft slightly positively buoyant before the mission began. The active ballast consists of tanks whose buoyancy could be modified during the course of a mission. It was used for in-mission system balance and buoyancy control. The ballast system was analyzed in order to aid in the placement of buoyancy tanks and weights on the model for the purpose of balancing the craft. Analysis was done manually using buoyancy equations. Although SolidWorks does not have a built-in buoyancy calculation tool, the buoyancy force is given by the weight of the water displaced by the model. The force would be found by changing the material of the parts of the assembly to water, and then using the mass properties function to find the total mass from which the weight could be found.

Drag. Since the estimated power required to appropriately drive WAVE in the forward direction was 10 Watts, a thruster pair capable of at least 10 Watts of thrusting power was needed. It was concluded that if the forward thrusters were capable enough to nominally drive the craft forwards, then those same thrusters could be used to move the craft along the different axis too. Several different potential thrusters were examined: trawling motors, commercial UUV thrusters, and modified bilge pump motors. In the spirit of keeping to COTS materials, custom built thrusters were only a passing consideration.

Trawling motors, although more than capable of supplying enough power, were found to be too heavy and power hungry. Weighing in on average at about 4 to 5 kg per unit and drawing up to 30 amps they were inappropriate for this application.

Figure 36: Trawling Motor (boxed) [54] [54]

Commercial UUV thrusters, such as the one shown in Figure 42, are ideally suited for the application seeing as that is what they were designed for. They are efficient and produce ample yet controllable amount of thrust. Unfortunately they are also expensive, at least $600 each. For this reason they were made inaccessible for regular use on WAVE. Two Seabotix BTD 150s were lent to the project by Navel Engineering Support Team (NEST) for the duration of the MQP which allowed for the exploration of thruster modularity.

Bilge pumps in and of themselves struggle to provide enough thrust simply by pushing water with their impellers, but if modified with a RC boat props can generate enough mechanical power to drive a small to medium UUV. A modified bilge pump is pictures in Figure 37. They also fall within acceptable electrical operational parameters and are naturally waterproof and are the most affordable to the three COTS thrusting systems. For these reasons they were selected as the most appropriate thrusters to use on WAVE.

Figure 37: Modified Bilge Pump

Tests were done to verify that the potential motors were fully suited for the application. Furthermore, different propellers were tested to try and optimize the efficiency of the thrusters.

4.5.1.1 Thruster Testing

These motors were subjected to thrust tests to give experimental motor data for better simulations of the design, and to determine if the thrusters would be sufficient to move the craft at 0.5 m/s. This was accomplished by doing static and flow tests in a tub of water using force-meters. The motor was strapped to a wooden rod, which was then strapped to a wooden board held in place on top of the tub. A force sensor was attached to the rod. This way, when the motor was turned on, it would exert a force on the rod, which could be read by the force sensor as a function of the moment created between the rod, which functioned as a pivot arm, and the force sensor. The set-up is shown in Figure 38.

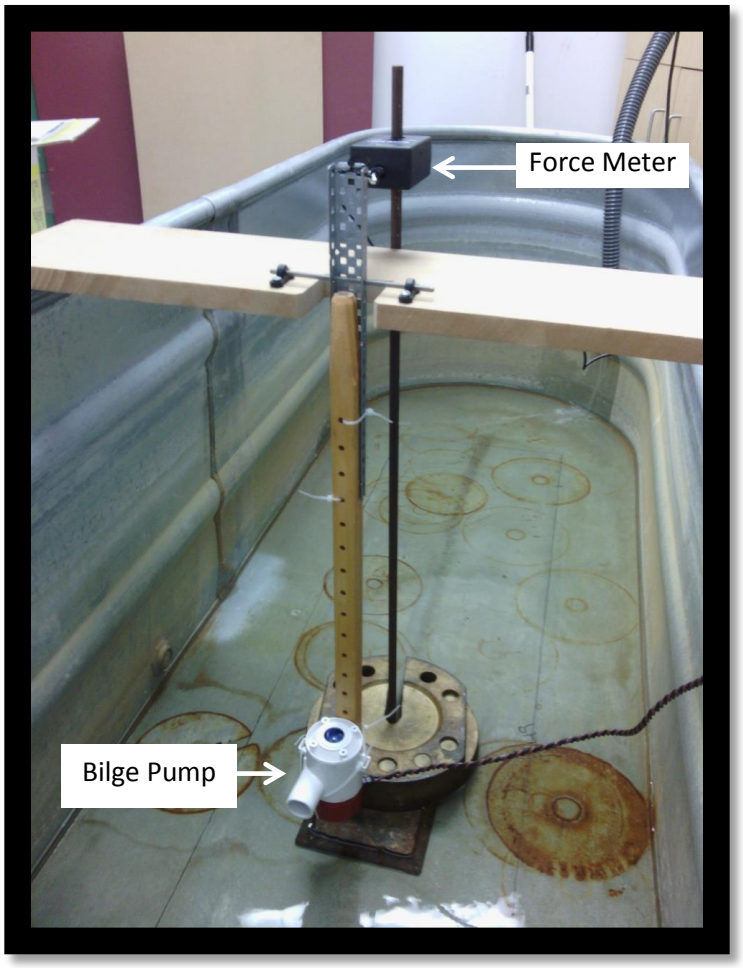


Figure 38: Thruster Testing Setup

Tests were run on a bilge pump before it was modified to fit a propeller. Force, flow rate, and power were measured from the bilge pump. The bilge pump was then modified to include a propeller. To do this, the white encasing (shown in the previous picture) was removed using a hand saw. After removing the encasing, a blue impeller was revealed. This blue propeller was removed to expose the bilge pump shaft. Next, the propeller needed to be attached to the bilge pump shaft which would be used in the final design. In order to do this a collar was fixed to the shaft with a screw. The propeller was attached to a screw and positioned inside the collar. This set-up stabilized the propeller to the bilge pump.

The graph in Figure 39 shows the mechanical power versus the electrical power for an unmodified bilge pump and a bilge pump modified for two different propellers.

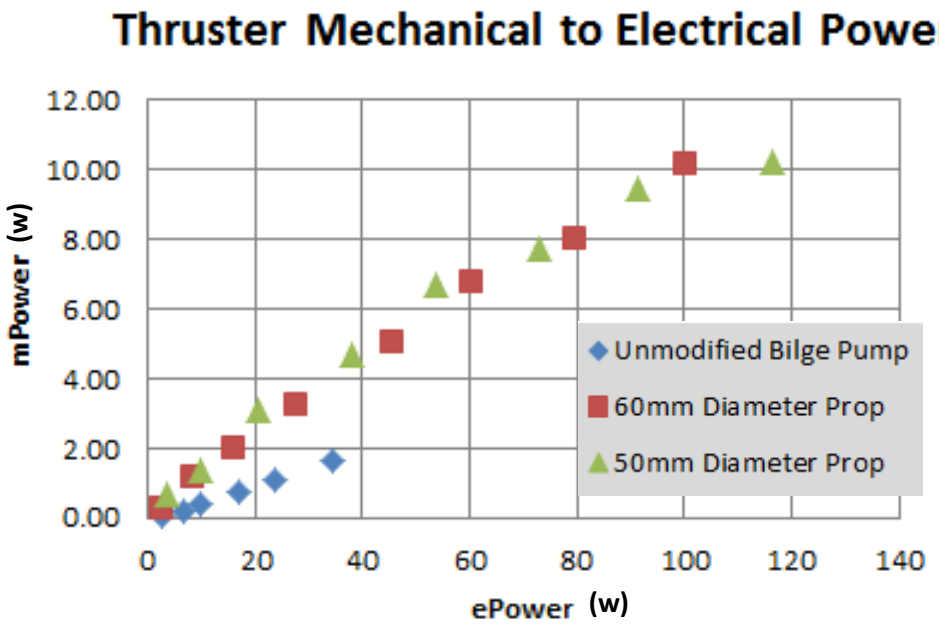


Figure 39: Graph of Bilge Pump Mechanical to Electrical Power

This graph shows the relations of the unmodified bilge pump, and the first and second tests on the propeller-modified bilge pump in terms of the mechanical versus the electrical power. Mechanical power was measured as flow multiplied by force. The electrical power was measured as current multiplied by voltage. This relation is meant to show the efficiencies of the motors. The graph shows it takes 100 W of electrical power to convert to 10 W of mechanical power, which shows the low efficiency of the motor.

The graph in Figure 40 shows bilge pump current versus voltage.

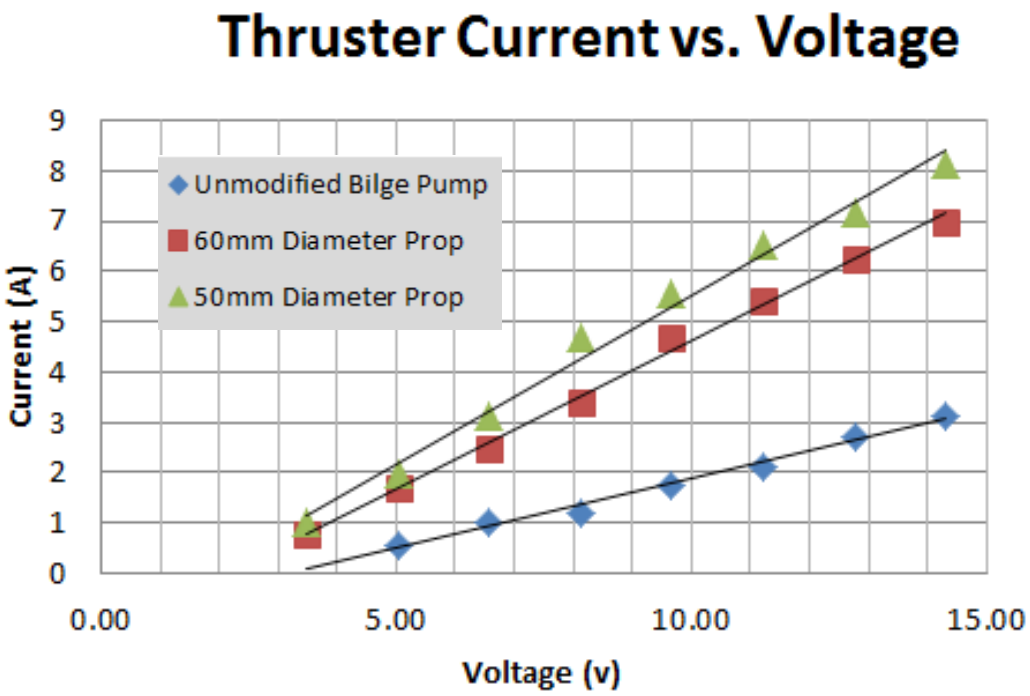


Figure 40: Graph of Bilge thrusters Current vs. Voltage

The graph shows the trend between the current and the voltage of the pump, and the first and second test done on the propeller-modified bilge pump. The thrusters were able to run continuously at 12 amps.

A test was done to determine the endurance of the motor. The purpose of this test was to measure the force generated by the pump over the length of a typical WAVE mission. The graph in Figure 41 shows the force generated by the bilge pump over the course of 20 minutes.

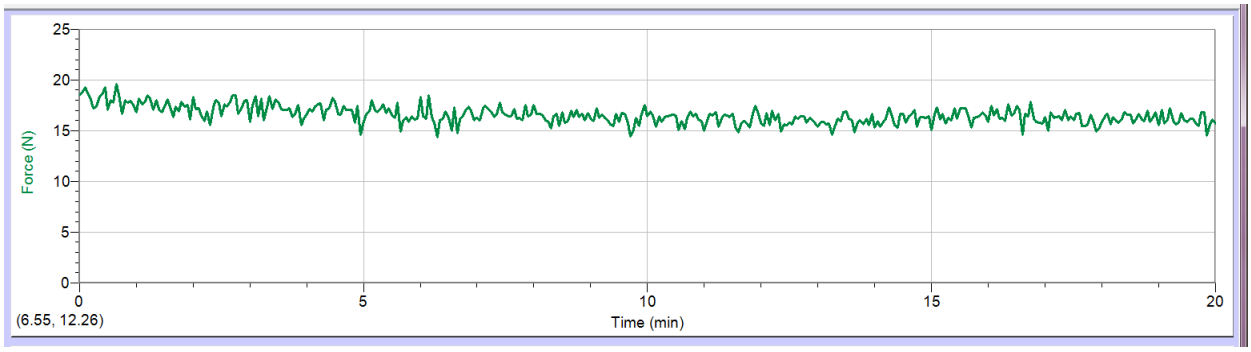


Figure 41: Bilge Pump Endurance Test, Graph of Force vs. Time

As shown in the graph, the motor maintained a relatively constant force over the course of 20 minutes. Several things were discovered about the bilge pump through these tests. These motors have low efficiencies. Also, after 20 minutes the bilge pump motors still maintained a steady thrust. This is important because it shows the motors are capable of running throughout the length of time needed for the AUVSI competition.

4.5.1.2 Specifications

The thrusters being used on WAVE are Seabotix BTD150 and Johnson Bilge Pumps.

The Seabotix BTD150 is pictured in Figure 42. Seabotix thrusters are recommended for small AUV operations. They have a nominal voltage of 12 V, and a maximum current rating of 4 amps. Each motor weighs 754 grams.

Figure 42: Seabotix BTD150

Following are charts showing data relevant to the performance of the Seabotix motor. Though nominal performance occurs as 12v, Table 6 shows how the motor performs at higher temperatures. Although the Seabotix motor can be run up to 6A, the life of the motor will drastically shorten.

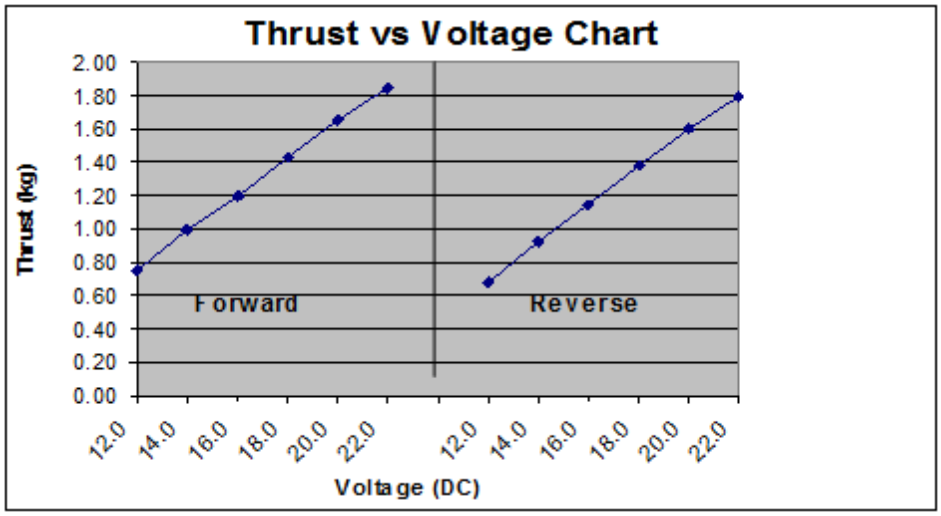


Figure 43: Graph Seabotix Thruster Data

Table 6: Interpreted Seabotix Results

Direction	Voltage(DC)	Current Draw (A)	Thrust (kg)
Forward	12.0	1.91	0.75
	14.0	2.45	1.00
	16.0	2.78	1.20
	18.0	3.32	1.43
	20.0	3.78	1.65
	22.0	4.20	1.85
Reverse	12.0	1.92	0.68
	14.0	2.42	0.92
	16.0	2.89	1.15
	18.0	3.30	1.38
	20.0	3.75	1.60
	22.0	4.23	1.80

The other type of motor being used on this submarine is a modified bilge pump. Two bilge pump models were considered: Rule 1100GPM and Johnson 1000GPM. Although the Rule pump, shown in Figure 37, was tested to be marginally more efficient, the Johnson pump, shown in Figure 44, was deemed easier to shroud for safety and was cheaper. Another benefit of the Johnson model was that it did not need modification other than adding a propeller whereas the Rule model had to have its shell removed through cutting.

Figure 44: Johnson Pump 1000 GPH

4.5.2 Safety Considerations

The primary safety consideration for these thrusters is protection from the propellers while the craft is in motion. This concern is addressed through shrouds, which form a protective shield around each propeller. The Seabotix motors being used are already fully equipped for underwater use. As such, their housing consists of a shroud to protect the propeller blades while the craft is in motion. The bilge pumps do not come with a shroud because they are built for use with an impeller, rather than a propeller, so

shrouds had to be constructed for them. The primary options for constructing these shrouds were to modify PVC piping or use rapid prototyping. PVC was chosen because it is significantly cheaper, although it requires more manual manipulation. The shrouds can be seen in Figure 45.

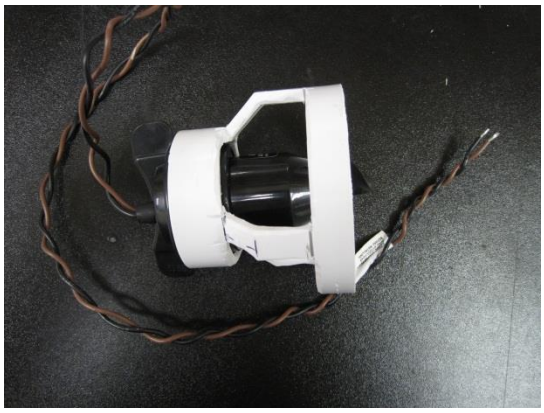


Figure 45: Propeller Shroud

4.6 Ballast

The ballast system needed to handle the load of all components of the base WAVE platform as well as most modules that might be added in the future with little to no redesign. To define the system, the overall buoyancy properties of WAVE were found by calculating the net buoyant force of each subsystem and finding the total buoyant force experienced by the platform. (Eq. 1), (Eq. 2), and (Eq. 5) were used with great frequency for this process as well as the specifications for each component, as is seen in Table 7.

Table 7: Buoyancy of Various Components

	Mass (kg)	Volume (cm^3)	Net Buoyancy (N)
Electronics Housing	18	12585	-53.121
Frame	1.86	1721	-1.364
Motors	2.294	1266	-10.062

Once the net buoyancy of the craft was found, a ballast system could be designed around needing 64.5N of positive buoyancy.

There are three potential options for the ballast system: a passive system, which utilizes buoyant foam and weights to achieve neutral buoyancy, an active system utilizing actuators to manipulate the buoyancy, and a hybrid design which uses a combination of the passive and active options. Each of these options was discussed and put into a design matrix for analysis. This design matrix can be seen in Appendix B: Ballast Design Matrix.

Table 8: Design Matrix Criteria

Criterion	Description
Cost	Overall cost both monetarily and in system resources.
Manufacturability	How easily the system can be manufactured.
In mission flexibility (trimming)	The range of control during a mission.
Payload range	The ability for the system to accommodate variable loads without extensive redesign.

The forms of ballast considered for control of the buoyancy were a fully active system utilizing a pump or piston, a passive ballast system utilizing buoyant foam, and a hybrid design using elements of both. After these options were compared through a design matrix comparing the options, the direct injection system combined with a small amount of buoyant foam was determined to give the most control for the least cost and complexity. It was determined with the equations above that the system needed to account for 68.6N of negative buoyancy. To do this, a system utilizing two Schedule 40, 10.16 cm (4") PVC tanks with lengths of 20.32 cm(8") and 0.007 cm^3 (427.6in^3) would supply the necessary positive buoyancy to counter the down force. Testing for the system was then performed. The PVC seals were tested for water-tightness, the pump was tested for positive displacement and reversibility, and the entire system was tested for controllability and speed.

The ballast system needed to be balanced to ensure uniform effect on WAVE, controllable via motors through the electronics board, and rigid for survivability in the different environments the platform will be deployed. The decision was made to use two reversible positive displacement pumps with a pair of PVC canisters. Each pump will be connected to a window motor to drive it, and the entire assembly is water proof, so no extra waterproofing is necessary. Each will be attached to the frame across from each other halfway along the length of the side bars on the top level of WAVE. They will have connecting tubes running to their own ballast tank, so each pump is only affecting one tank at any given time.

There are two individual ballast tanks vertically attached to either end of the frame, front and back.

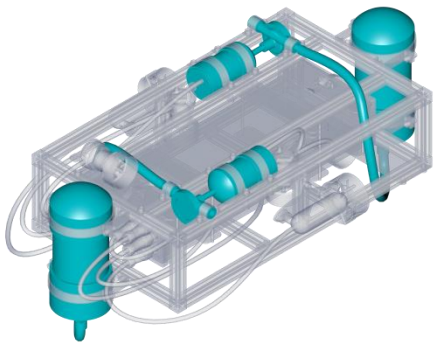


Figure 46: Ballast placement

They will both be connected via 3/4inID tubing to the pumps with tight seals to keep from leaking. The vertical tanks will provide stability for the water within the tanks to reduce sudden pitch changes as the water moves, and they will allow for control of overall robot pitch. The tanks were designed to only need 2.5 to 5 cm of water to bring the craft to neutral buoyancy.

PVC was chosen because it is lightweight and durable. It can also be customized to be different shapes as needed and is cheap compared to materials such as acrylic and Delrin. Further, the material

property data of PVC is widely available, making research and calculations of weight and buoyant properties easier. For the tanks, the following calculations were used to determine the minimum buoyant force supplied by the system:

$$Total\ F_g = mg$$

$$m = V_{PVC} * \rho_{PVC}$$

The density of Schedule 40 PVC is 1.41g/cm^3.

$$m = (V_{OD} - V_{ID}) * \rho_{PVC}$$

$$m = \left((32.6612cm^2 * \pi * 20.32cm) - (25.8064cm^2 * \pi * 20.32cm)\right) * \frac{1.41g}{cm^3} = 617g$$

$$F_g = 6.1N$$

$$F_B = 2084.43cm^3 * \frac{9.81N}{kg} * \frac{0.001kg}{cm^3} = 20.45N$$

$$F_T = F_B - F_g = 14.35N$$

The above value is for only one tank. There will be two tanks, as well as 7.007*10m^3 (427.6in^3) of buoyant foam, attached to the WAVE platform. The buoyant foam's properties are shown below;

$$\rho_{Foam} = \ 57.666\frac{kg}{m^3}$$

$$F_{Net}\ per\ cubic\ cm = \ 0.0093\frac{N}{cm^3}$$

$$V_{req} = \frac{93.7N - 28.7N}{0.0093N/cm^3}$$

$$V_{req} = \ 6989.25cm^3$$

The pumps were selected for their ability to fill the task of being reversible pumps that had the potential to move water quickly in and out of the ballast tanks. The pumps are rated at 852 liters (225 gallons) per hour, though that number is more than enough to fill the needs of the robot given that the amount of water needed to be moved is 0.95 liters(0.25 gallons). At peak performance, the pump can fill the tanks to the required level in 4 seconds. Peak performance of the pumps is achieved at 1200rpm, but the pumps on WAVE will be run at 150rpm. This means that the tanks will take roughly 30 seconds to fill completely to the point of having negative buoyancy equal to that of the positive buoyancy with empty ballast tanks. The pumps weigh only 226.8g each and have a net buoyancy of each pump is positive 0.185N.

5 Electrical Design and Analysis

The electrical infrastructure of WAVE requires coordination between several robotic subsystems. The electrical system needed to be modular in order to allow for future expandability. Multiple hardware and software configurations were considered when deciding on the modular infrastructure. Additionally, computational and form factor requirements were factors that drove the design process. The powering and programming of this system were important processes for the electrical team. During the design process, four requirements were taken into consideration. These requirements were that WAVE must have a modular infrastructure, support low-level drivers, implement a reliable power distribution system, and interface with a wide range of sensors. The specifications for the system, as highlighted in previous sections, were that the max-open loop voltage had to be 60V, and the power system had to be able to source at least 700W. These specifications were used to complete the design and meet the system requirements. In this section, the processes for deciding all aspects of the electrical system are outlined.

5.1 Modular Infrastructure

Most robots only have one specific directive, whereas WAVE is designed to be able to have a variety of tasks to execute. Currently no off-the-shelf electronics exist to implement this degree of modularity, so a new electronics system had to be constructed to achieve the desired result.

The modular infrastructure that powers WAVE had to meet a series of basic requirements. The system had to be easy to use, utilize standardized communication protocols, and be highly scalable and parallel. Ease of use does not necessarily relate directly towards modularity, but promotes widespread adoption of the system powering WAVE for use by future projects. The use of industry standard communication protocols such as USB and Ethernet made the infrastructure compatible with a wide range of existing hardware, leading to more modularity. Using standard protocols also reduced the cost of the system and the development time. Scalability is important to the modular infrastructure because

if WAVE was to be modular, the platform needed to be able to accommodate a multitude of accessory modules.

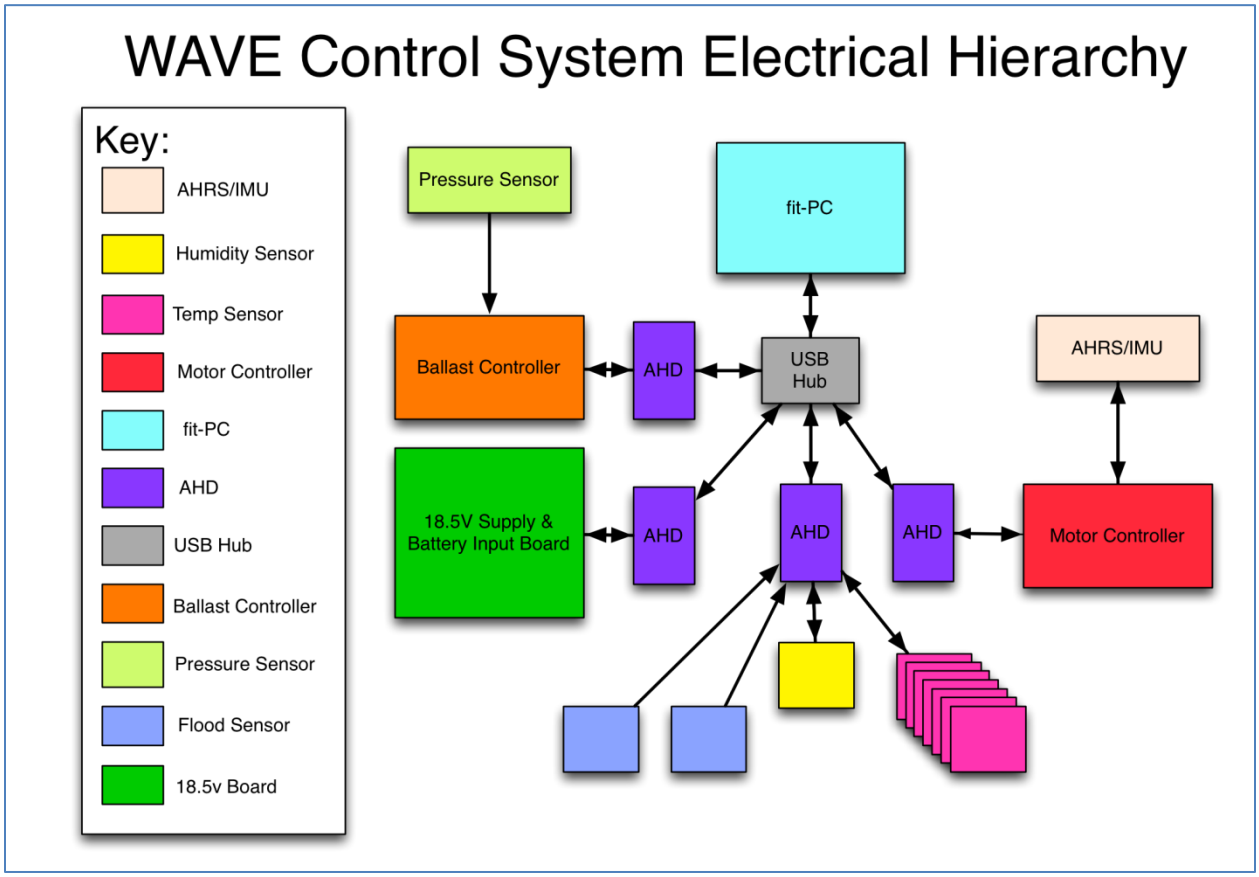


Figure 47: WAVE's Modular Infrastructure

Parallel processing is essential to a modular system. Each module has a dedicated processing unit, so that modules will not interfere with each other in terms of processing load. The combination of parallel processing and scalability means WAVE gains more power with the addition of more computing modules. Although these additional modules can add additional power and cabling requirements, the computational performance gained by adding more processors in parallel are important to consider. Parallel processing is the ability to carry out multiple operations or tasks simultaneously, and the ability to capitalize on this capability was crucial.

In order to implement this modular system for WAVE a custom embedded computing device was required. The custom device needed to be an abstract piece of electrical hardware that could apply to various applications. This custom embedded computing platform was called the Abstract Hardware Device (AHD). When implemented, the modular infrastructure was composed of four AHDs which all serve specific functions to give WAVE basic functionality, more AHDs can be added to expand WAVE's capabilities.

5.2 Abstract Hardware Device Design

The first major challenge of designing and implementing the modular infrastructure for WAVE was the design of the AHD. With the electrical structure being the core of WAVE's modularity, a parallel processing platform was required. This platform needed to be highly scalable and easy to use. Through analyzing existing microcontroller boards, a list of requirements for WAVE's modules was determined. The design analysis yielded the following requirements:

- High Speed Serial Communication over Standard personal computer interfaces e.g. Ethernet &USB
- Motor Control Capabilities
- Compatibility with a large range of both analog and digital sensors,
- General purpose digital IO capabilities
- Real time closed loop control implementation
- Cost effective design
- Integration with a large variety of external add-on hardware

The requirements for the microcontroller board were determined based on the basic requirements for the modular infrastructure and the requirements that all robotics oriented system controllers must implement. No current modules existed that would provide the future extensibility

desired for this project. Therefore, the decision was made to create the Abstract Hardware device from scratch.

The next step after establishing basic requirements for the Abstract Hardware Device was to choose the hardware that would power it. Choosing hardware for the AHD first required establishing concrete specifications for the AHD itself. The first requirement for the AHD was to be able to communicate with PCs at a high speed. A central processing unit was necessary for WAVE's design, so the AHD needed to consider this fact. For more information on WAVE's distributed processing, refer to 6.1. With today's computers the most popular serial interfaces are USB and Ethernet. USB was the first choice for a serial link due to its high frequency of use on PCs; however, USB does have constraints. USB's most relevant constraint is data bandwidth. If this system was to be truly modular, the infrastructure could not be limited by bandwidth. Ethernet is another very common and powerful communication interface. Ethernet is a widely used standard with bandwidth limitations far exceeding that of USB. Another less relevant constraint of USB is that it is limited to 4.9m (16ft) cable length in other applications (non AUV) that constraint could be problematic. Ethernet solves that problem with the ability of transmitting upwards to 304.8m (1000ft) cable lengths. Since both USB and Ethernet are industry standards the decision was made to include both on the AHD.

Another specification that needed to be accounted for by the AHD was the ability to drive motors. Motor drivers and controllers are commonly controlled using a pulse width modulation signal (PWM). Driving the motors is an important task for the AHD to complete, and PWM signal generation is the most widely accepted standard for accomplishing our desired goal. Thus, the decision was made to account for PWM signals when designing the AHD. Rather than bit-bang a signal, the AHD required hardware PWM generation. It is important for the AHD to be able to generate all PWM signals using discrete hardware instead of with software via bit banging because software PWM can hinder

computational performance as well as using up all available onboard timers of the processor that could be better used for more important control system functions.

Also, the AHD had to be able to interface with sensors and additional hardware. Conventionally other controllers accomplish the task of sampling sensors and interfacing with additional hardware with the following onboard circuitry: General purpose digital IO, analog-to-digital converters, and low level primitive serial communication. General purpose digital IO (GPIO) is used for interfacing and controlling digital hardware. GPIO is handled by the digital pins or general purpose pins on a given controller. GPIO pins read in high or low signals and output high or low signals; this process is used to read digital sensors and trigger digital hardware. Analog sensor sampling is accomplished by an analog-to-digital converter (ADC). An ADC converts a range of analog voltage to integer numbers that the controller uses within its on board program to accomplish its task. ADCs are the only way to interface with analog sensors; therefore, having ADCs built into the controller was important for the AHD. Aside from interfacing with analog and digital sensors the AHD needed to be able to interface with additional hardware that doesn't connect over analog or digital IO. To accomplish the task of hardware expansion the AHD needed to feature some of the more common low level serial interfaces such UART/USART, SPI, and I2C.

5.2.1 Microcontroller

Now that the embedded computing requirements for the abstract hardware device have been established, the next step in development was to choose the controller (microcontroller) that would power the AHD. Microcontrollers from a range of manufactures were examined to find one that would fit our list of requirements. Only two were initially deemed acceptable, AVR and PIC32.

While a lot of the AVR models (particularly the AVR32 models) had all the desired features, they had a few problems. The AVRs were very expensive: most of the models that had all the desired features cost around 11-15 USD in quantities under 1000. For the purpose of this project, that price range was

unacceptable. In addition to being costly, the AVR32 has a max clock speed of 66MHZ. Since the AVR32 was based on out-of-date hardware architecture, this family of microcontrollers would not be relevant to our hardware needs. One positive feature of the AVRs that was very appealing was a large open source software community with out of the box usable software libraries that would be helpful for implementing the AHD. One of these libraries was the LUFA stack, the lightweight USB for AVR stack. This USB stack is known for being highly efficient with minimal latency, a solution to a problem we will have to deal with in the future. In the end, however, the cons of the AVR outweighed the pros and we went on to examine additional controller families.

The PIC32 chips have a high clock speed, low cost, all the serial peripherals we needed, and a nice set of developer tools as well as already supporting the Neuron Robotics Bowler protocol (one of our potential software solutions). Unfortunately the PIC32 had too many deal breakers. The PIC32 suffers from closed-source driver software that Microchip controls. This is a problem because if one wants to utilize the full processing potential of the PIC32 they need to purchase Microchip's optimized compiler software to have all the driver code run at full speed. While Microchip offers a free non-optimized compiler, the differences in performance are far too large to merit using the PIC32. Without the optimized compiler things such as USB and Ethernet would be plagued with inconsistent latency issues, something that would negatively affect the overall performance of the modular system.

The AHD required a lot of features, and neither the AVR32 nor the PIC32 could meet every requirement. In fact none of Atmel's and Microchip's offerings could satisfy the stringent hardware requirements demanded by the Abstract Hardware Device specification. From researching what was new in the microcontroller market, the reality was discovered that almost every major IC manufacturer including Atmel was moving to ARM based solutions. In the industry ARM processors are gaining popularity in embedded applications. ARMs are usually known for being used to run small embedded

operating systems for simple computer devices such as cellphones and digital media devices. In recent years there was the birth of a new generation ARM, the Cortex M ARM. The Cortex M series come in the following configurations: M0, M0+, M1, M3, and M4. The numeric value on the M scale correlates to clock speed features and mathematical computation capabilities. After researching, the conclusion was reached that NXP Semiconductor had the best offering of ARM Cortex M series controllers. NXP's offerings were very cost effective, filled with features, and had excellent documentation, high clock speeds, open source software libraries, outstanding tech support, and friendly developer tools. After searching through NXP's catalogue and comparing the information with all of the criteria outlined above, a Cortex M4 was decided to be the most appropriate controller for the AHD. Another feature that was discovered shortly after finding Cortex ARMs was that NXP had ported over the LUFA stack to their processors, having support ready to go for all their Cortex M3 and M4 models. This fact was the selling point that made NXP's Cortex M4 implementation win over the other chip families. In the end the NXP LPC4330FBD144 was chosen for having all of the serial protocols needed: ADCs, the LUFA stack for efficient USB applications, a 204MHZ clock speed, an onboard asymmetrical M0 processing core at 204MHZ, and a complete repository of example code and comprehensive documentation.

5.2.2 Abstract Hardware Device

The physical layout of the Abstract Hardware Device is outlined in the following section. The decisions behind the pin layout, schematic, signals, and integrated design are explained in further detail. These four items describe the core characteristics of the Abstract Hardware Device.

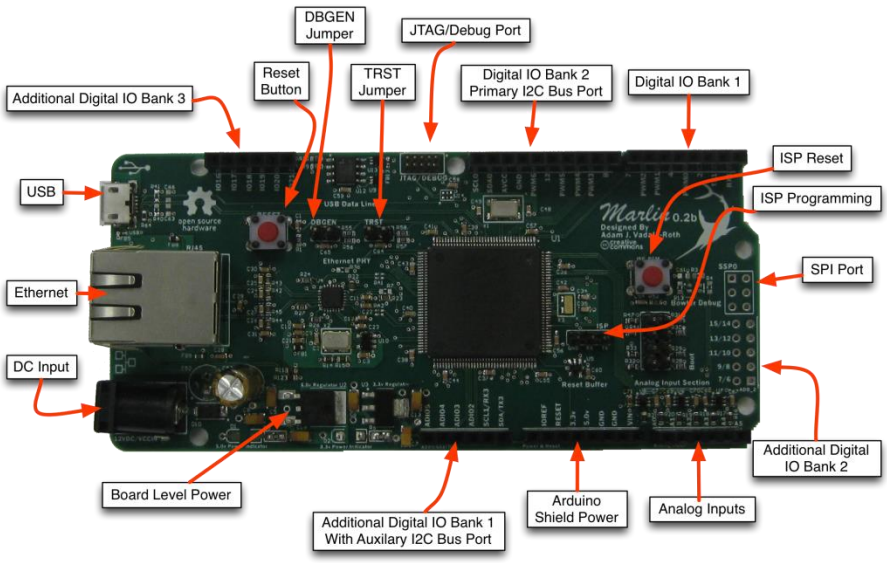


Figure 48: Abstract Hardware Device

5.2.2.1 Pin Layout

Once a controller had been decided upon, the next step was to begin the design of the AHD. First off, the pin-out of the AHD needed to be decided. The pin out was important for additional hardware to attach to the AHD and provide modularity. Initially, a standard single row of pins was favored as plugging into a breadboard for prototyping would be easy. After more research, the idea of using the Arduino shield layout was brought up. The Arduino layout has compatibility with a large range of off-the-shelf hardware that would hasten module development as well as being a popular open source hardware standard. Ultimately the Arduino layout was chosen for compatibility with off the shelf hardware. With less additional hardware needed to develop, the process would be better. However, when time came to design the circuit board for the AHD, the physical form factor of the Arduino was not used. The form factor of the Arduino was too small to fit all of the USB, Ethernet, and supporting circuitry for the NXP chip. Instead of designing an entirely new form factor with Arduino pin out, the AHD was designed to implement the form factor of the Racal Micro development board, which is a longer board shape that already had the Arduino headers. Going with an existing open source form factor helps simplify the design process. The Arduino Leonardo pin layout can be viewed in Figure 49.

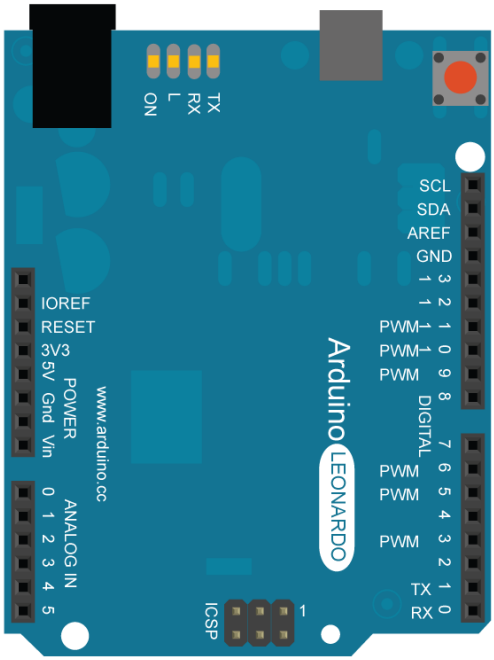


Figure 49: Arduino pin layout [55]

5.2.2.2 Schematic Design

For the design of the schematic and circuit board for the AHD, the program Altium Designer 10 was selected for an intuitive user interface and advanced routing tools. During the process of creating the schematic for the AHD, the LPC4330FBD144 was discovered to lack the necessary number PWM outputs. To solve this problem, a coprocessor based on the ATMEga328MMH was added. A coprocessor was chosen over converting GPIO pins into PWM outputs using an internal clock module because it posed no negative impact to the processing core. The ATMega328MMH was chosen over other microcontrollers because it consumes minimal power, has a complete prewritten code library for all required functions, and the development tools were inexpensive compared to other offerings. The schematic for the board was created primarily from the reference design provided by NXP with some slight modifications to make sure everything would function correctly. For a more detailed look at the Abstract Hardware Device reference Appendix I: Accessing Board Schematics

.1 .

5.2.2.3 Signals

When the schematic was complete the next step was routing the signals on the board itself. Routing for the board was very straight forward when it came to the Ethernet section. The Ethernet traces had to follow strict electrical standards. The RX and TX lines needed to be routed as differential pairs. In order to produce a working Ethernet circuit, the reference design for the Ethernet circuitry was consulted and followed strictly. In a case like this one, the best method is to stick to manufacturer reference designs to avoid bug-ridden or non-working circuits. The Ethernet section was traced almost identically to the reference design with respect to the Rascal board form factor and surrounding circuitry.

Every section of the board design was retraced about three times. The main challenge in PCB design is choosing the orientation of the controller and where all the complementary circuitry is placed. After experimenting and retracing, the perfect placement and pin mappings were determined. Then the entire board was retraced a final time and read for fabrication. The process of getting the board fabricated is a challenge in its own right. Traces, holes and vias had to be adjusted several times until they met the manufacturer's machine specifications. Working with the fabrication company and their online design checking tools, all the errors were eliminated over the course of a week and the board was ordered. This first design was the prototype of the AHD, and multiple revisions were needed to attain maximum functionality.

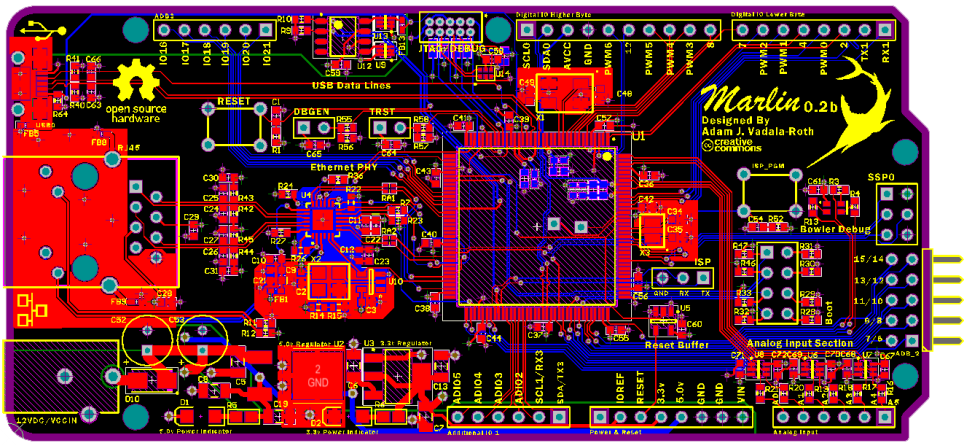


Figure 50: AHD Signal Traces

5.2.2.4 Integrated Design

After the design and prototyping period of the project the much discussed Abstract Hardware Device (AHD) is finally a reality. The final AHD module manifested as a custom printed circuit board (PCB) based around the NXP LPC4337JBD144 ARM Cortex M4F micro-controller chosen during the design and methodology process. The PCB provides the LPC4337JBD144 with all of its required supporting circuitry and all the AHD specific hardware peripherals and interfaces. The supporting circuitry includes a board level power supply , JTAG programming interface, Real Time Clock (RTC) crystal oscillator, system clock crystal oscillator, in system programming (ISP) interface, boot source jumper select, reset circuitry, analog power conditioning, and bypass capacitors on every voltage pin. The power supply consists of a 3.3 and 5 volt rail; each rail of the supply is damped, decoupled, and filtered to assure consistent voltages for stable operation of the LPC4337JBD144. The JTAG programming interface consists of the 10 pin ARM Cortex M JTAG/Debug port , debug enable (DBGEN) pin jumper circuit, and the JTAG test reset (TRST) pin jumper circuit. The crystal oscillators consist of a separate crystal for both RTC and system clock and their associated load capacitors, these oscillators provide the LPC4337JBD144 ARM processing core and real time clock peripheral with the required clock signals they need to function. The ISP programming interface consists of a 3 pin header wired to RX/TX of the LPC4337JBD144's UART1

interface and to ground as well as a ISP reset button. If the user does not wish to use a JTAG programmer they can load on firmware over the ISP interface by holding down the reset to put the LPC4337 in ISP program mode. The reset circuitry consists of a reset button for pulling the reset pin low, a 10k pull-up resistor (bypassed with a 100nF cap), and a logic buffer to prevent ESD from resetting the LPC4337JDB144. The analog power filtering section connects the onboard ADC's analog power input, and analog ground to 3.3v power plane and the ground plane through ferrite beads. The boot source jumper select consists of a 2X4 pin header. One set of 4 pins are wired to the boot select pins on the LPC4337 and the others are wired to ground; the boot select jumpers are used for choosing a where the LPC4337 will boot from. Lastly every pin on the LPC4337JBD144 that receives power from the 3.3v power plane has a bypass capacitor to ensure stable voltage is constantly supplied.

The AHD specific peripherals and hardware interfaces are what make the AHD what it is and not just a simple microcontroller breakout PCB. The onboard peripherals for the AHD include, a 10/100 Ethernet transceiver and jack, a USB 2.0 port, and overvoltage protection. The hardware interface is how the AHD is expanded into an application specific module for WAVE. The AHD hardware interface consists of Arduino Leonardo compatible pin headers and three additional AHD specific headers for module applications that require more digital inputs than what is available on the Arduino headers. Through the hardware interface the AHD achieves Arduino Leonardo hardware compatibility and functionality with the only difference of being 3.3v compliant on all digital pins. In order to achieve this sort of compatibility an analog signal level/range conversion section was added to the analog input header to achieve 5 volt sensor compatibility.

5.3 Embedded System Design

The modularity of WAVE required a well thought-out embedded system design. A development environment needed to be determined. The functionality of this environment also needed to be confirmed. Communications also needed to be established between AHDs, sensors, and the Fit-PC. The

components of the embedded system design can be classified into two distinct groups: serial communications and other peripherals. The layout of WAVE's embedded software architecture can be seen in Figure 51: Embedded System Architecture

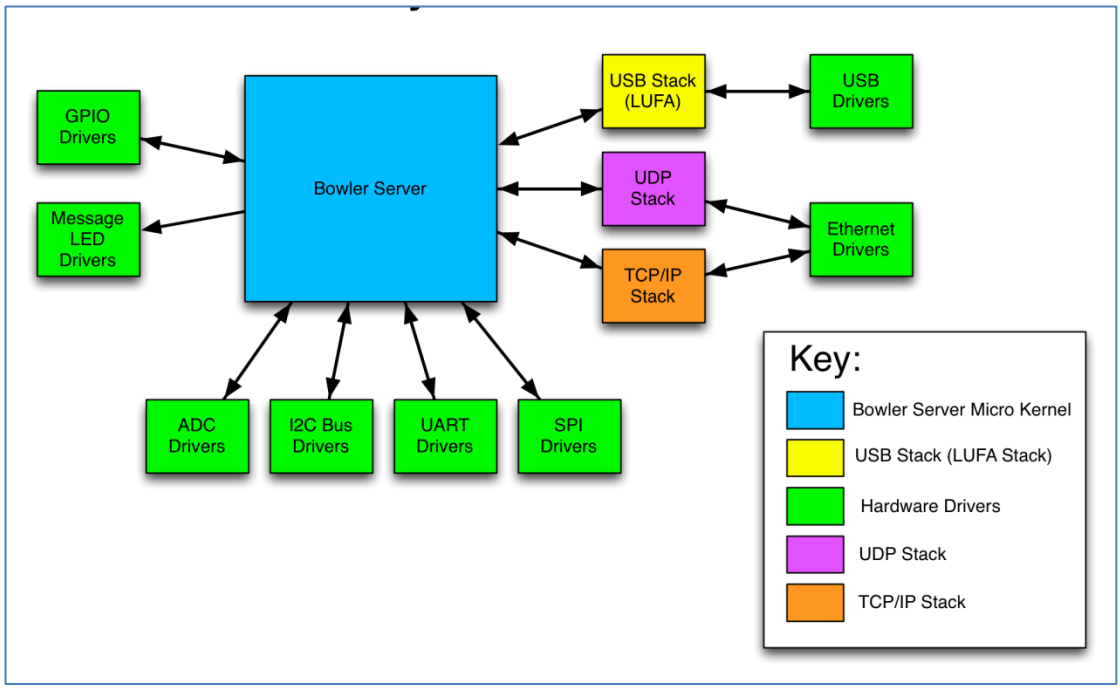


Figure 51: Embedded System Architecture

5.3.1 Development Environment

In order to access the modular infrastructure on the Abstract Hardware Device, embedded computing had to be implemented to ensure low-level functionality. All of the embedded programing was written in the C language. The software was developed in the LPCXpresso programming environment provided by NXP and Code Red Technologies. In this environment, Neuron Robotics' Bowler protocol was ported over to interface with the NXP-provided libraries. This combination would successfully allow for all of the low-level functionality of the Abstract Hardware Device to be configured and controlled from the top level program.

5.3.2 Functionality Confirmation

Before writing unique functions, two sets of example code were implemented to prove functionality. First, a series of LPCXPresso development tests were used. These examples confirmed that the development environment was set up correctly. Additionally, they proved that code could compile correctly with the driver libraries and that the functionality of multiple peripherals including LED, UART, Ethernet, and Audio could be ascertained.

Once these tutorials were proven to be successful, another series of example projects were tested. Specifically, we utilized NXP's Lightweight USB Framework for AVRs (LUFA). The LUFA stack enables communication over USB, which allowed all of the modules to interface with each other and the central processing unit. The LUFA stack came with several example projects to test communication over USB. LUFA can be utilized as a device, a host, or both. The successful implementation of these tests proved this functionality. Once communications had been handled at the embedded level, functions could be designed and implemented for the peripherals. Each module had methods that needed to be defined at the higher level in Java, but before those functions could be implemented, basic functionality needed to be established at the embedded level. Afterwards, the higher level methods had the green light for design.

5.3.3 Serial Communications

The serial communications design encompasses how different module hosted on an AHD would communicate with each other as well as with the Fit-PC. Four serial communications standards were used in the design of the embedded system. These standards are USB, Ethernet, SPI, and I2C. USART/UART was designed to implement SPI and I2C protocols. Having a wide range of communications standards is necessary because different sensors communicate using different formats. Therefore, to fulfill the goal of making WAVE a truly modular development platform, a wide variety of communication

protocols were planned. Even if certain protocols were not used in this iteration of the project, future teams may need these communication standards.

5.3.3.1 USB

USB is the primary means of serial communication in the WAVE embedded system design. Each module was designed to be able to send data to and receive data from the fit-PC. USB 2.0 technology was preferred over USB 3.0 in this design because of the widespread acceptance of 2.0 as well as the team's inexperience with 3.0. The Bowler Communications protocol was designed to be ported over to the AHDs and monitored through USB serial communication. The port was designed to be completed over USB, and progress of moving the Bowler firmware was to be monitored through the USB echo server. While each AHD has the means of communicating via all of the serial communication peripherals, USB is the only peripheral that is actually implemented by all of the modules at this time.

5.3.3.2 Ethernet

Ethernet is a secondary means of serial communication in the embedded system design. The Ethernet design is in a preliminary state. Since USB was designed to be the primary means of serial communication for the first iteration of WAVE, the software design for Ethernet was not as in-depth. Ethernet was still deemed necessary because future teams may want to communicate over a UDP stack, so the technology was included in the design of the embedded system. Room exists for future teams to expand the scope of Ethernet support in the AHD.

5.3.3.3 USART/UART

USART and UART are both devices that can implement asynchronous serial communications. Both contain two wires. One wire is the transmitter (TX), and the other one is the receiver (RX). Both UART and USART implement asynchronous modes. Both USART and UART were necessary for the design of the embedded system because asynchronous transmission may be needed for different AHDs to

communicate with each other. While WAVE currently does not implement USART or UART communication, the ability for future development necessitated the incorporation.

5.3.3.4 SPI

SPI is a daisy-chainable 3 or 4-wire serial interface. These wires include one for clock synchronization between the two communication devices, sending data in sync with the clock wire, receiving data in sync with the clock wire, and the chip select which is used to enable a slave device connected over the previously mentioned wires. SPI is very popular in industry and necessary for the design of the embedded system. Different AHDs need to be able communicate in synchronous modes, which SPI enables. The sensors on this iteration of WAVE do not require SPI communication, but different sensors with similar capabilities require SPI. Additionally, the ability to daisy-chain different modules and stipulate which ones are slaves and which one is the master is an important feature in this design.

5.3.3.5 I2C

I2C is a two wire serial interface that is also daisy-chainable. One wire is for transmission and receiving and one is for clock synchronization. For a low-level serial interface I2C is fast as well as being easy to use from a programming standpoint. Some of the sensors used in WAVE communicate using I2C, so this standard was incorporated in the design of the embedded system and deemed critical.

5.3.4 Other Peripherals

The design of the other peripherals stipulates how the data sent via serial communications is obtained. While they are linked to communications their design is a separate process. These other peripherals are ADC, SCT, and GPIO.

5.3.4.1 ADC

Some of the peripherals and sensors generate an analog signal as their output. Therefore Analog-to-Digital Converters are necessary components of the embedded system design. The ADCs will convert

these analog signals to a digital data. The digital input can then be sent to the Fit-PC or other modules via serial communications.

5.3.4.2 SCT

In addition to supporting a wide range of serial communication protocols the controller must have a state configurable timer (SCT). A SCT equates to high speed mathematical computations which are essential for implementing real-time closed-loop control. A State Configurable Timer allows for real-time closed-loop control. Real-time closed-loop control is an essential application for the AHD, so a controller with a SCT was a must.

5.3.4.3 GPIO

General Purpose I/O pins can be configured as either inputs or outputs. GPIO pins provide a flexible infrastructure while reducing costs. These two facts are responsible for the inclusion of GPIO in the embedded system design. When set as an output, the values of these signals can be changed to either high or low. When set as an input, the states of these signals are read through GPIO control register bits. Additionally, GPIOs can be used as status indicators (for example, with LEDs), which is another important piece of the system design.

5.4 Sensor System Design

For the sensors sub-system design, the chosen components would sense the following parameters: inertial measurements, compartmental flooding, temperature, humidity and depth sensing. The sections below detail the decisions that went in to deciding upon the specific components for the design.

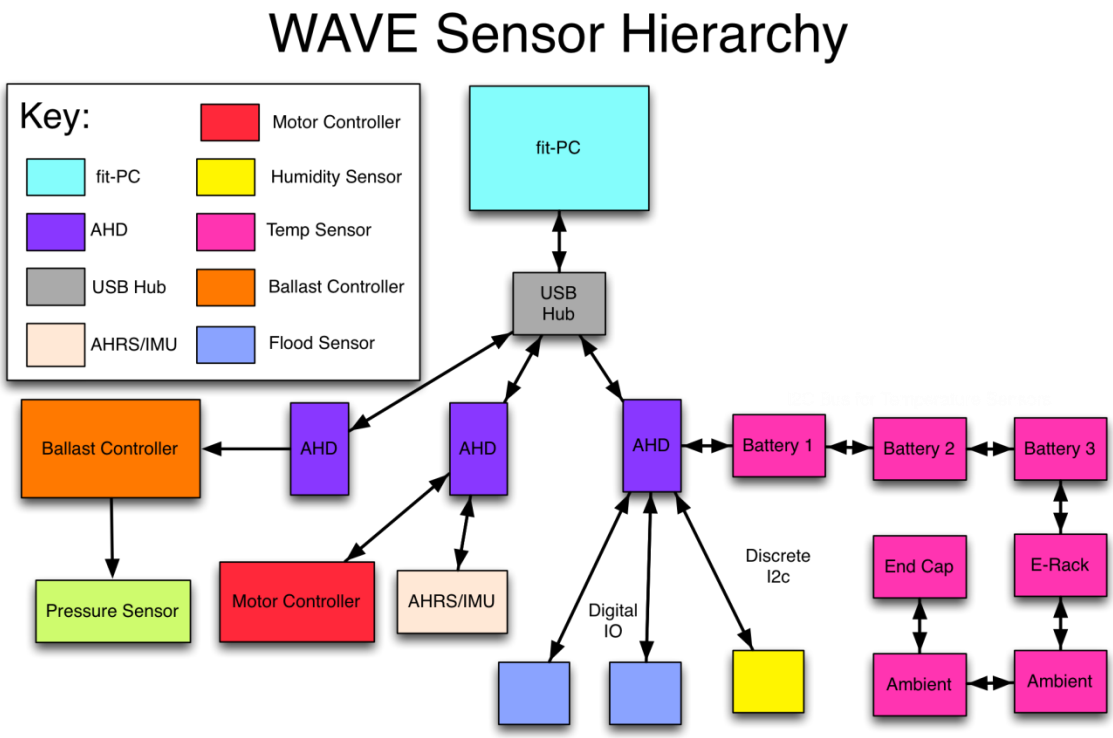


Figure 52: Sensor Suite

5.4.1 Inertial Measurement Unit

In deciding on an inertial measurement unit, the factors considered were: functionality, pricing, and user interface. The first model examined was the SparkFun Digital IMU Breakout - IMU3000. This model was initially chosen due to the fact that this sensor has both an accelerometer and gyroscope allowing us to measure both acceleration and angular acceleration. In addition this device had programmable sample rates for the gyroscope and accelerometer which could be used with the modular infrastructure. The decision was made not to proceed with this device because it contained a gyroscope and a breakout in which an external accelerometer would be attached. In addition this device did not include a magnetometer or compass. In conclusion the device was too basic for the tasks required of the inertial measurement unit.

The next model considered was the Atmel ATAVRSBIN2. This inertial measurement unit has 9 degrees of freedom, and also contains an accelerometer, gyroscope, and magnetometer. The rates at

which acceleration and angular velocity are measured are programmable. This ability is ideal for allowing future teams to change the rate at which acceleration and angular velocity are measured depending on the environment of the robot. The Atmel ATAVRSBIN2 has an I2C interface for all three sensors and produces a digital output which when sent will give us data responding to the orientation of the WAVE modular system. In addition to these functionalities this IMU also has a temperature sensing unit that may be used in order to measure the internal temperature of the pressure vessel. This device did contain all the necessary functionalities, but the decision was made not to go forward due to the fact that the device would not provide the necessary level of accuracy and precision. The level of precision required of an IMU is based on how fast of a rate the dynamics of the craft change. WAVE's dynamics as it moves underwater would be changing very slowly. In a dynamic system with a slow rate of change the IMU data needs to be sampled over a longer period of time. Sampling from an IMU over a longer period correlates to more error in the data. If WAVE is to navigate reliably an IMU with a much lower drift is required so that data can be sampled at the appropriate rate for WAVE's change in dynamics without inducing too much error.

Figure 53: Microstain 3DM GX3-35

Ultimately, the Microstain 3DM GX3-35 (shown in Figure 53: Microstain 3DM GX3-35) was decided upon. This model includes many of the same features as the Atmel ATAVARSBIN2. As such this

model allows for programmable rates which allow for the user to change the measurement scales upon the different sensors contained in this module. This model contains a tri-axial accelerometer, tri-axial gyro, and a tri-axial magnetometer. This model measures with greater accuracy and precision compared to the Atmel ATAVARSBIN2 and other low end IMUS. In addition this model has also been successfully used in other under water platforms. For the purposes of the project, this module will be powered via USB and calibrated to be read and display measurements onto the GUI developed by the software team. A side-by-side comparison of features and price is shown in Table 9: IMU Comparison Analysis.

Table 9: IMU Comparison Analysis

Sensor	Price (to team)	Accelerometer	Gyroscope	Magnetometer
Sparkfun Digital IMU	39.95	N	Y	N
Atmel ATAVARSBIN2	45.16	Y	Y	Y
Lord Microstrain 3DM-GX3-35	0	Y	Y	Y

5.4.2 Depth Pressure Sensor

In selecting a pressure sensor the factors that were considered were: price, measurement range, and durability. The first pressure sensor that was considered was the Honeywell – PX2AN2XX150PSCHX. This pressure transducer can measure between 0 - 6.8 atm. This transducer has an operating voltage of 5V, producing an analog output corresponding to the depth. This pressure sensor was a good choice although is listed to be optimal in the following environments: HVAC, Air Compressors, and light hydraulic system.

Figure 54: GE PDCR 1830

As such, the decision was to use the GE PDCR 1830(shown in Figure 54: GE PDCR 1830) which is a depth and liquid level sensor that outputs a voltage between 4 mv and 20 mv. The optimization of this sensor for water environments led to choosing this sensor. This sensor can measure between 0 – 1.02 atm. The millivolt output needed to be amplified using an instrumentation amplifier before being read into an ADC. This device is powered by 10 V and as such will implement a buck converter in order to step down the voltage from the 12 V power source. The comparison of these two modules is displayed in Table 10: Depth Sensor Comparison.

Table 10: Depth Sensor Comparison

Sensor	Price (to team)	Operating Range	Suitable for Underwater
Honeywell PX2AN2XX150PSCHX	89.48	0-6.8 atm	N
GE PDCR 1830	0	0-1.0 atm	Y

5.4.3 Liquid Level Sensor

The team considered two varieties of liquid level sensor when designing the sensor suite. The first of these was the Basement Watchdog Water Alarm, a commercially available product, for use in your home. Powered by a 9V battery this sounds an alarm when the sensor probe touched water. This sensor

could be removed from the Basement Watchdog Water Alarm, in order to create a flood sensor that fit the needs of the robot. As such the decision was made that it would be preferable to buy a sensor that already provides this functionality because the cost of components would outweigh the cost of a sensor. In keeping with this the Honeywell LLE102000 Liquid Level Sensor was selected. This sensor uses a supply voltage of 5V and provides a digital output corresponding to if water is or is not detected within the system. This sensor detects water when the dome is fully submerged in water at a depth of 22.49 mm. This system would be optimally designed to contain flood sensors in as many points to compensate for any shifting of the robot while under operation. It is necessary that the flood sensors be located at least near the end-caps which are the first entry point for any water leaking into the pressure vessel. As such this system was designed in order to include at least two flood sensors located at the two ends of the pressure vessel. Although, at this stage of the design only one flood sensor was implemented within the pressure vessel due to size constraints. For this implementation the flood sensors was mounted dome down near the center of the pressure vessel although for optimal implementation it is necessary that the flood sensors be mounted dome upwards to function properly also that for future implementations the flood sensor be located at the four corners at the vessels to account for any tilting and also along the horizontal sides of the vessel as well.

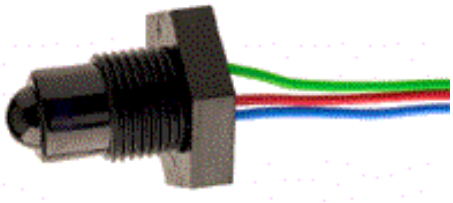


Figure 55: Honeywell LLE 10200

5.4.4 Temperature and Humidity Sensors

It was also necessary to detect temperature and humidity. In order to do so temperature sensors were employed in order to detect the temperature of different components. A humidity sensor was chosen in order to detect the ambient humidity. In considering temperature and humidity sensors they were

chosen based off sensing capabilities within the electronic pressure vessel. An outline of these capabilities is outlined in Table 11: Temperature and Humidity Sensor Comparison.

Table 11: Temperature and Humidity Sensor Comparison

Sensor	Price (to Team)	Temperature and Humidity Sensing	Number of Sensing Channels
Humirel HTM1735LF	27.28	Both	Ambient Temperature Ambient Humidity
Microchip TC-74	11.84	Temp Sensing Only	Up to 8 Locations
Sparkfun HD100	9.95	Humidity Sensing Only	Ambient Humidity

The first sensor examined was the Humirel HTM1735LF temperature and humidity sensor board. This board has sensors to detect both temperature and humidity, containing a negative temperature coefficient thermistor and a relative humidity module. This model was initially chosen for cost efficiency and implementation in previous designs. Although later in the project, the need to sense temperature at multiple locations was discovered. This capability could not be easily accomplished using the Humirel board, therefore a change was made to the Microchip TC-74 to sense temperature and the Spark fun HD100 in order to sense humidity. The Microchip TC7-74 is supplied with 5V and is capable of sensing temperatures between -40℃ and 125℃. This temperature sensor has is relatively accurate $\pm$ 3 ℃ at temperatures between 0 ℃ and 125℃ and between $\pm$2 ℃ for temperatures lower than 0℃ . The Spark Fun HD100 is capable of detecting humidity between 1 and 99% relative humidity. Both of these devices communicate via the I2C interface. All of the temperature and humidity measurements will be communicated via one I2C bus line. Temperature is sensed in seven different locations, and ambient humidity is sensed.

5.5 Power System Design

In order to ensure modularity, WAVE required a power distribution design capable of sourcing high power with multiple voltages available. The batteries and voltage rails needed to be determined, as well as how to implement a safe and reliable system.

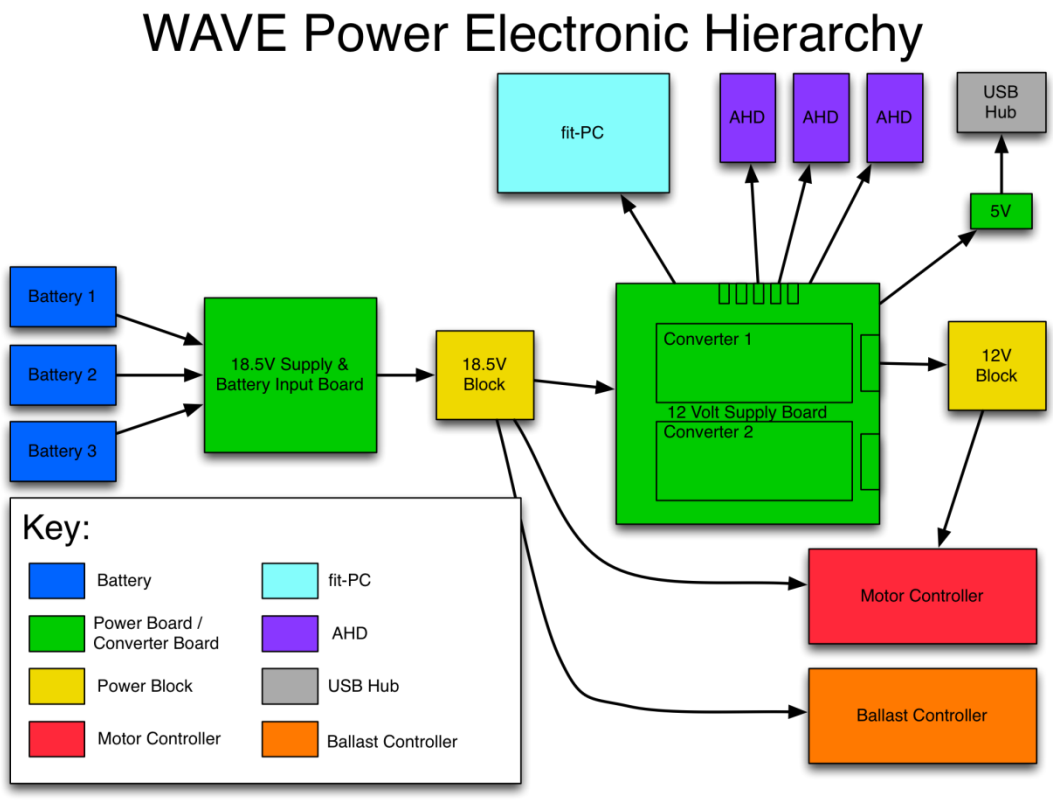


Figure 56: Power System

5.5.1 Power Distribution System

As all subsystems were simultaneously being created, it was necessary to determine power constraints in order to successfully design WAVE's power distribution system. With this, the mechanical engineering team was given the constraint to choose thrusters that operate between 12 and 18.5 volts (V) nominally, as research and experience have shown these to be typical motor values. Additionally, the custom AHDs and computer were given the constraint to operate off 3 amps (A) or less as experience had shown this to be sufficient power for the application. From this, a worst-case, estimated power

budget was created based upon what WAVE might contain and was used as a foundation for starting the power system design. This initial estimated power budget is shown in Table 12. Note that this was the preliminary power analysis for the system, which was updated as power specifications were determined from the other subsystems. The final power budget is shown in Table 13.

Table 12: Estimated Power Budget

Component	Max Current(A)	Nominal Voltage(V)	Max Wattage(W)
Thruster 1	6	18.5	111
Thruster 2	6	18.5	111
Thruster 3	6	18.5	111
Thruster 4	6	18.5	111
Thruster 5	6	18.5	111
Thruster 6	6	18.5	111
Buoyancy Actuator 1	4	12	48
Buoyancy Actuator 2	4	12	48
Buoyancy Actuator 3	4	12	48
Buoyancy Actuator 4	4	12	48
PC	3	12	36
AHD with Motor Control Shield	3	12	36
AHD with Onboard Sensor Shield	3	12	36
AHD with Power Management Shield	3	12	36
AHD with Buoyancy Control Shield	3	12	36
AHD with Emergency Protocol Shield	3	12	36
AHD with External Sensor Shield	3	12	36
6 Additional Modules (1 AHD each)	18	12	216
Total	**99**		**1326**

With this information, the power source had to be able to safely supply 99A at different voltages which totals to 1326W. WAVE's power source selection began by examining lithium polymer batteries. As discussed in section 2.6.4.4, LiPo batteries are known to be light in weight and high in energy density in comparison to other battery chemistries, which was why they were examined first. The highest capacity lithium-polymer battery available for off-the-shelf purchase was 10Ah. To find the worst-case run time use this equation:

$$\left(\frac{BatteryCapacity}{MaxCurrent}\right) * CapacityDischarge = WorstCaseRunTime$$

where CapacityDischarge is assumed to be 70% and is the amount of the battery's energy used before turning the system off to recharge the battery for the next use. (This is especially important for lithium polymer batteries, because cells discharged under 3.0V can lose their capacity and degrade the overall battery performance.) With one 10Ah lithium polymer battery, the worst-case run time would be 4.24 minutes; however, the system requires a 20 minute run-time as shown in Table 2 of section 3.2.

This short run time brought up the consideration of connecting two 10Ah lithium polymer batteries in parallel for a total of 20Ah, resulting in 8.48 minutes for the worst case run-time. Though it is possible to connect lithium polymers in parallel, it can be very dangerous, especially with such large batteries. Given the volatility of lithium polymers batteries, there was apprehension about connecting them in parallel so this consideration was put to the side. The possibility of using a safer, more familiar battery chemistry that could still source the current, but with a higher capacity was then considered.

Lead acid batteries filled the requirements, but by this time in the design process, two thrusters were chosen that run off a nominal voltage of 19V. To power these thrusters, two 12V lead acid batteries would need to be connected in series, which would give us a 24V output. The 10Ah lithium polymer battery weighs 1.2 kg, in contrast to only one lead acid battery weighing around 3.3kg. Though lead acid batteries meet the power specifications, they would add an additional 5.4 kg to the weight of the chassis. That weight would have to be counteracted by other buoyant components or by constant actuation by the vertical thrusters. "LiFePo" batteries were also considered, but an off-the-shelf unit that met the current specifications was not found. Two LiPo batteries onboard were again considered, but with one connected solely to the motors, the other connected solely to the electronics; however, the team was not enthusiastic about this and wanted a system that shares an equal load and discharges evenly. An integrated chip produced by Linear Technologies was found with the ability to automatically

switch back and forth between two batteries, giving the run-time capacity of two batteries in parallel but without them actually being connected. This would have given us the desired safety and capacity, but was not implemented due to how close it was found to the deadline of a finished power system.

The initial idea to have LiPo batteries in parallel was revisited. An RC battery management unit (BMU) was found and additional research found many successful examples of batteries connected in parallel when particular LiPo safety guidelines were followed; these guidelines can be found in Appendix F. With these various options and with extensive modularity in consideration, the decision was finalized to create the power system with the ability to be powered by up to 3 LiPo batteries in parallel.

The battery that was chosen is a 5S, 18.5V, 10Ah, 25C, lithium polymer battery made by Gens ace. This means that each battery is comprised of five individual lithium polymer cells connected in series. An individual cell when fully charged is 4.1V and should not be discharged below 3V. The voltage of each cell in series adds to create a total nominal voltage of 18.5V, but will be 21V when fully charged. The battery can source 10 amps continuously for 1 hour before needing to be recharged again and the "C rating" of 25C indicates that it can source 25 times the amount of capacity it has; therefore, it can safely source 250A.

To ensure system safety during every WAVE mission, each battery has its own BMS connected that monitors the voltage of each individual cell in the battery. It connects directly to the battery's balancing plug and if a voltage difference of 0.2V is found between any two cells, a network of resistors is used to slowly balance the battery pack. Also, Schottky diodes were implemented to keep batteries from charging or discharging each other and safety fuses were also included in the design.

As mentioned previously, the power budget was adjusted as design decisions were made and the finalized budget can be seen in Table 13.

Table 13: Final Power Budget

Component	Max Current (A)	Nominal Voltage (V)	Max Wattage (W)
Seabotix BTD150	5.8	19	110.2
Seabotix BTD150	5.8	19	110.2
Bilge Pump	6	12.5	87.5
Bilge Pump	6	12.5	87.5
Bilge Pump	6	12.5	87.5
Bilge Pump	6	12.5	87.5
Window Motor	2.5	12	30
Window Motor	2.5	12	30
Fit-PC	1.5	12	18
AHD Actuator Control Shield	1	12	12
AHD Onboard Sensor & Safety Shield	1	12	12
AHD Thruster Control Shield	1	12	12
AHD External Sensor Shield	1	12	12
2 Additional Modules (1 AHD each)	2	12	84
USB Hub	4.9	5	24.5
Total	**53**		**828.9**

According to these updated values, Equation 1 shows the worst-case run time with 1 LiPo battery connected to be approximately 8 minutes long. The average run time for 2 batteries is estimated to be around 20 minutes, but this depends on the type of mission deployed.

5.5.2 Voltage Rails

Due to the modular nature of the system, it is important to have industry standard voltages available for ease and flexibility of device implementation. For lower current devices, such as sensors, 3.3V and 5V

are common voltage specifications. Also, many other devices, such as motors, operate on 12V. Therefore, a 12V rail was designed to power each AHD which will then have its own 3.3V and 5V rails onboard through voltage regulators. As deciding the best type of battery and configuration for the system took longer than anticipated, designing a high current voltage regulator became impractical. With this, a Vicor DC/DC converter was found that will convert the unregulated 18.5 nominal voltage from the battery into the 12V rail and can source 200W at 16.67A continuously; however, this does not entirely fulfill the requirements of the system. This is one of the highest powered DC/DC converters found with an efficiency of 85%, so two of these converters were implemented.

5.5.3 Battery Monitoring System

As previously mentioned, lithium polymer batteries can be dangerous if not given proper care and attention. Each cell in a healthy battery should discharge evenly, but in the case that one cell is bad and discharges more quickly, it will continue to do so until the battery is ruined or even combusts. If a lithium polymer cell is discharged below 3V it may be permanently damaged and not maintain a charge afterward. Also, if a battery becomes too hot, it may suffer from thermal-runway resulting again in a ruined battery or combustion. Therefore, it is very important to monitor the current being sourced by the batteries, the voltage levels, and as common practice, the temperature – all in real time.

Initially, voltage and current sensors made specifically for lithium polymer batteries by RF-Flyer were going to be daisy-chained and implemented for this purpose. They connect directly to the batteries' balancing plug, allowing each individual cell within the battery pack and the total current being sourced to be directly monitored by the fit-PC. This was the plan until the lead time on the current sensor was discovered to be impractical for the design deadline. Also, the decision to connect batteries in parallel while implementing the BMS for additional safety removed them as an option as the BMS also connects to the balancing plug and cannot be daisy-chained.

The system was then redesigned to incorporate four Allegro Hall-effect current sensors; three to sense the current of each battery and the fourth to sense the power tether. Desiring as little voltage loss as possible, the voltage sensing was decided to be accomplished through an isolation unity gain amplifier monitoring the main 18.5V rail. This draws a negligible amount of current and will have a voltage divider on the output that steps the voltage down to a value between 0V and 3.3V and passes it to the analog to digital converter, where 3.3V at the ADC input corresponds to 21V at the battery. More information regarding the voltage sensing can be found in section 8.2.2. A temperature sensor will be placed directly on each battery pack for monitoring and was discussed in section 5.4.4.

As the batteries are able to source 250A continuously, the current is not being monitored solely for battery health, but is also used to ensure proper system functionality. For instance, if two batteries are connected in parallel and one is sourcing more current, this shows that the system should be examined. Along with the voltage sensing, the current sensing is also a way to monitor the amount of the battery that has been discharged. It is recommended that lithium polymer batteries be discharge to around 70% of the capacity or less before ending the mission and charging again. When the battery is fully charged it has a voltage of 21V and a capacity of 10Ah; when discharged it should never drop below 15V without a load or below 7Ah. Therefore, WAVE has the ability to monitor the battery capacity by sampling the amount of current at a given time interval and subtracting it from the 10Ah capacity, by monitoring the voltage level, or both. Also, the temperature of the batteries should never exceed 120°F. Should any of these limits be exceeded, the design called for the fit-PC to turn off the motors and surface as described in section 4.3.6. For more information on how to safely use LiPo Batteries, reference Appendix E1. How to Safely Use Lithium Polymer Batteries

5.5.4 Power Tether

The tether system was to be used as an option for testing and consists of two 14 AWG modified power cables, one of 3m and the other of 30m. The length used depends on the type of testing to be done. The

shorter cable allows for more power to be transferred whereas the longer cable allows for more mobility. Table 14 accounts for the voltage drop through the cable by showing the voltage that WAVE would receive depending on the current and length of the cable

Table 14: Tether Resistance and Voltage Drop

Cable Length	Resistance of Cable	Voltage In (20A)	Voltage In (50A)
3m	0.025756	23.48	22.71
30m	0.25756	18.85	11.12

A suitable AC power supply, or a pair of regular automotive lead acid or marine batteries in series for a total of 24V can be used as the tether power source. An optional regulator could be added inside WAVE to create a more stable input voltage and a more controlled testing environment. The tether and tether power source was not purchased as it was not a priority for WAVE, but it is something that can be implemented in the future and was accounted for in the design.

5.6 Design of Printed Circuit Boards

With any printed circuit board design there's a strict set of guidelines to ensure that all electrical signals can travel their desired paths without interference, ensure minimal crosstalk between adjacent signals, ensure a robust mechanical structure, and to ensure proper functionality of active electrical components. There are several basic principles to keep in mind when designing a printed circuit board. The components must be laid out in such a way that straight traces can be used. Right angle traces should never be used. Instead, two forty-five degree angles should be utilized. Signals between layers should never be overlapped. Power supply traces should be kept away from analog and digital signals. All connections to common ground should follow the shortest path of inductance (e.g. PCB traces are essentially inductors, the lower the inductors the better). Finally, there should always be a continuous

ground plane beneath analog and digital signals. An important decision is what design paradigm should be used – either the maze or X/Y paradigms should be used.

5.6.1 Power Board

In order to distribute the amount of power required by WAVE, custom printed circuit boards that can safely source high power were made. First, understanding of high power PCB designing restraints was required. The power supply PCB was first made as one board, but was later divided into two separate PCBs to better meet standardized sizes, which results in a lower manufacturing cost. The two boards are the main power supply board and the power conversion board. See Appendix I: Accessing Board Schematics for more details and schematics of the board.

5.6.1.1 Designing High Power PCBs

When designing a PCB for high current and high voltage it's best to stick to the basics to start but in the end high current design requires a few more rules to follow. When transmitting high current power through a PCB standard trace widths fewer than one hundred thousandths of an inch are not sufficient. When designing high current PCBs the same concept of requiring thicker wire for high current power transmission applies it is not as simple as "thickening the traces". In order to achieve the same properties of thick copper wire traces must be made wider or instead replaced with copper pours. A copper pour is polygon drawn in the PCB design software and will later be made out of copper when the PCB is manufactured. By making polygons with a significantly larger area than normal sized traces the volume of copper that the signal will be running through is greatly increased. Just like thicker wire has a larger volume of copper than thinner wire and therefore can carry more current the same applies to PCBs. If a PCB meant for high current power transmission were to be designed using standard width traces, there would be many problems such as overheating of nearby components due to increased heat in the PCB, thermal damage to the PCBs materials, and in the worst case burning of the PCB or traces exploding. When designing the power distribution PCBs for WAVE for both the 12v supply board and

the 18.5v battery input board all of the traces for the 18.5v power rails were constructed from copper pours. On both supply board the paths in which high current power travels are made from copper pours; any other signals, sensor or low voltage supplies were routed using standard sized traces following the design rules of basic PCB design outlines previously discussed.

5.6.1.2 Main Power Supply Board

The main power supply board can be seen in Figure 58 consists of current sensing, voltage sensing, signal conditioning, and outputs a nominal 18.5V rail which is the main voltage rail of the system. The board has four input connection points. Three of the input connection points are designated for the three batteries to be connected in parallel and the fourth connection is designated for powering WAVE through a tether. Each input has its own safety fuse, Schottky diode, LED for indicating battery or tether presence, current sensor, and voltage sensor consisting of an op amp and a voltage divider. The current and voltage sensors go into a Microchip MCP3208 SPI analog to digital converter which connects to an AHD for monitoring the power system. The power traces of each input converge, creating the main rail, which has a network of filtration capacitors. Following the capacitors is a large resistor that drains the capacitors after the system is turned off. The discharge rate of the capacitors can be seen in Figure 57. The capacitors are discharged to less than 1V after 30 seconds and are completely discharged after 7 minutes and 45 seconds. This prevents capacitors with stored energy from accidentally being shorted and injuring the user.

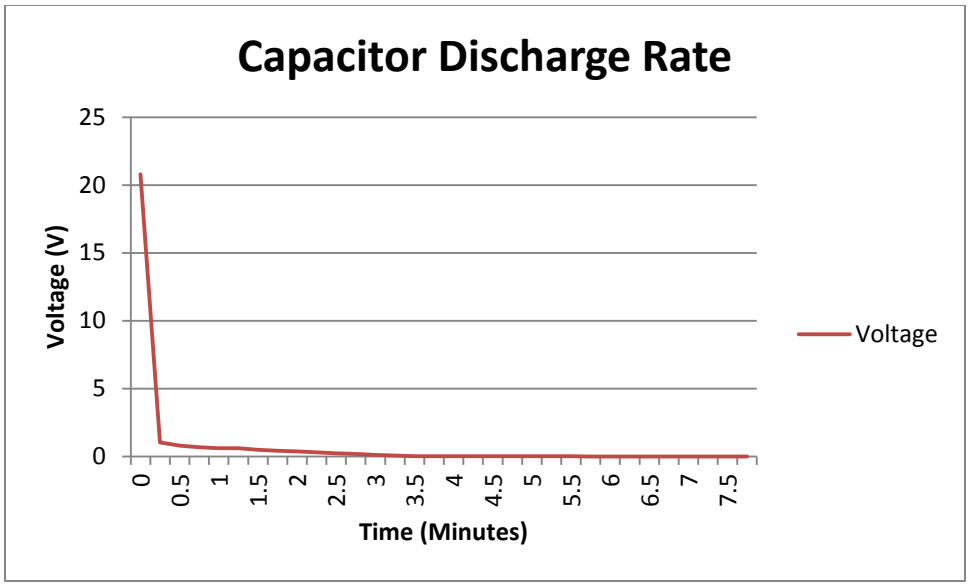


Figure 57: Capacitor Discharge Rate

The main rail had an output which then went into the system's on and off switch. In this iteration of WAVE, the switch was located inside the electronics housing. Rectifying this design should be a high-priority task for future teams. From the switch, the main rail goes into a distribution block which then drives the motor boards and the power conversion board.

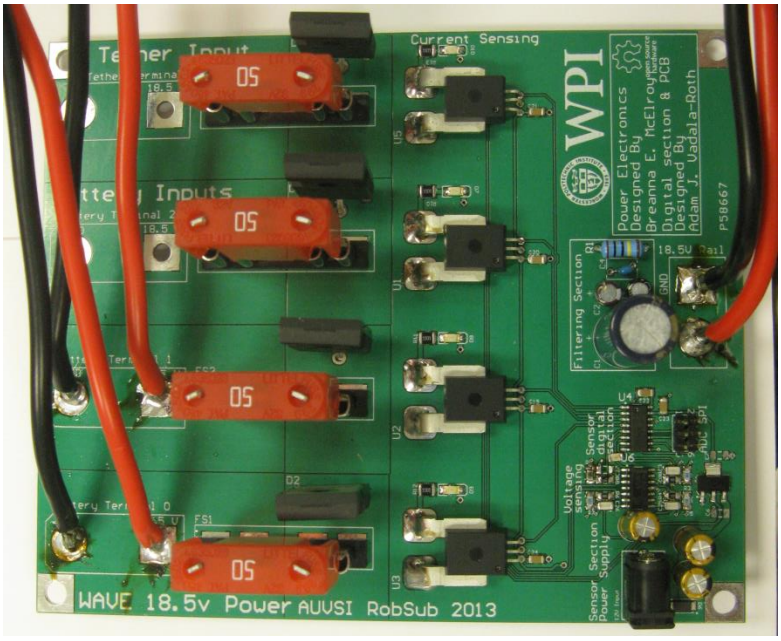


Figure 58: Main Power Supply Board

5.6.1.3 Power Conversion Board

The power conversion board, as can be seen in Figure 59, converts the 18.5V main rail into 12V which powers the AHDs, the fit-PC, and any other 12V component that may be needed in the future. Note that the conversion board had to be modified, as is discussed in section 8.2.2.4.1, so some components are on the other side of the board. The conversion board consists of two 12V Vicor converters which each have their own series of bypass and filtering capacitors. The board takes in the 18.5V rail and outputs two different 12V rails (one from each Vicor converter). One 12V rail is designated to power all AHDs and the other rail is designated to power the fit-PC. The converters are limited to 200W each; this is why two converters were implemented, providing a total of 400W available for WAVE's modular system. Additionally, a Pololu D15V70F5S3 5v DC/DC converter was connected to the screw terminal on the 12v conversion board for powering the on board USB hub.

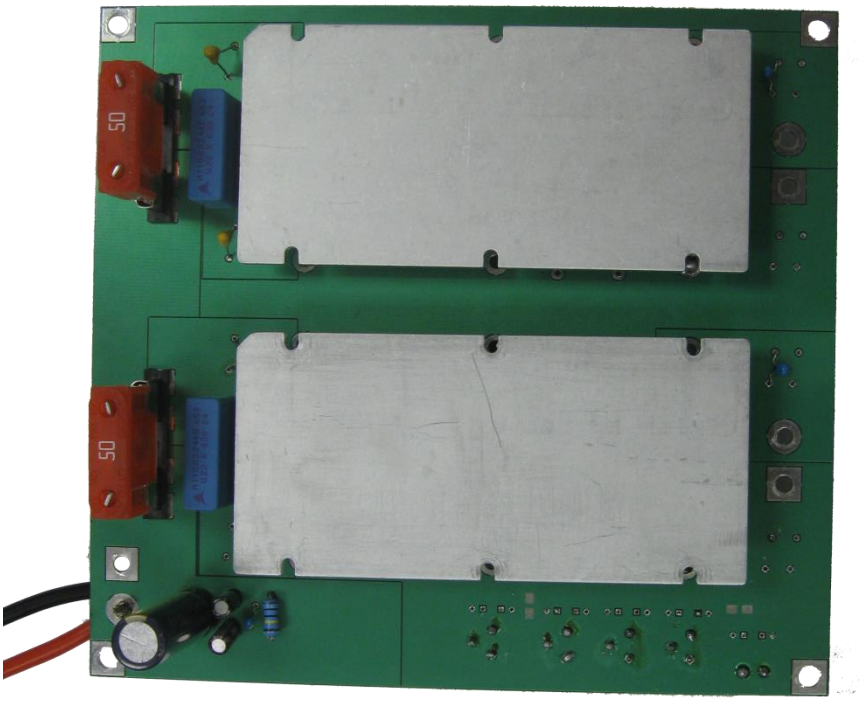


Figure 59: Conversion Board

5.6.2 Motor Board

The thruster controller for WAVE is a fairly simple design. The thruster controller designed for WAVE is based upon the two channel high current motor shield made by Pololu Robotics using the same STMicro VNH5019 high current H-bridge. The controller was realized by creating a PCB design containing six VNH5019 H-Bridge ICs each with their own power filtration, charge pump switching FET, and inline signal resistors (for 3.3v and 5v logic compatibility). To keep the design and operation organized, the H-bridge ICs were grouped into three pairs, mirroring the configuration of three pairs of thrusters on WAVE's chassis. For the H-bridge circuitry the design reflects the Pololu shield and the STMicro datasheet schematic identically but some modifications were necessary in order to make this system appropriate for WAVE. It is not possible to control these 6 H-bridges with the standard Arduino

Leonardo pin out, so for the controller each pair of H-bridges have their control signals connect to a pin header; the thruster controller interfaces with an AHD via a custom made shield that connects to the three headers on the thruster controller via ribbon cables. Another feature that is unique to the thruster controller design is an improved current sense section. Onboard the PCB is a Microchip MCP3208 SPI ADC used to sample the current sense signal from each H-bridge closest to the signal source as possible; this method was chosen in lieu of sending the signal over the ribbon cables where the data could degrade. The ADC connects to the AHD via the 6 pin SPI port where the ICSP port would be on an Arduino Leonardo board.

5.6.2.1 Thruster Controller & Navigation Shield

Another challenge posed by this project is a challenge many robotics engineering projects need to overcome, controlling actuators. In order for WAVE to move it needs six thrusters; in order for those thrusters to work they will need a controller. The thrusters chosen by the mechanical design team consist of a combination of two SeaBotix BT150D thrusters with the remaining four being comprised of Johnson bilge pump cartridges with model boat propellers attached.

The six chosen thrusters are brushed DC motors, this is important because a controller for a brushed DC motor is significantly different from that of a brushless motor. Brushed DC motors can be controlled with a single half bridge or a single H-bridge. An H-bridge is a switching circuit that consists of four switches wired together to form an H shape; between the four switches the direction of the current running through the motor can be switched allowing control of the motor in both directions. A half bridge is only half the amount of switches as an H-bridge and only allows control of the motor in one direction. Brushless motors would require a much more complex circuit consisting of multiple half bridges for controlling the individual phases. Since the thrusters chosen for WAVE were built around brushed DC motors a motor controller consisting of H-bridge integrated circuits was needed. The thruster controller board can be seen in Figure 60.

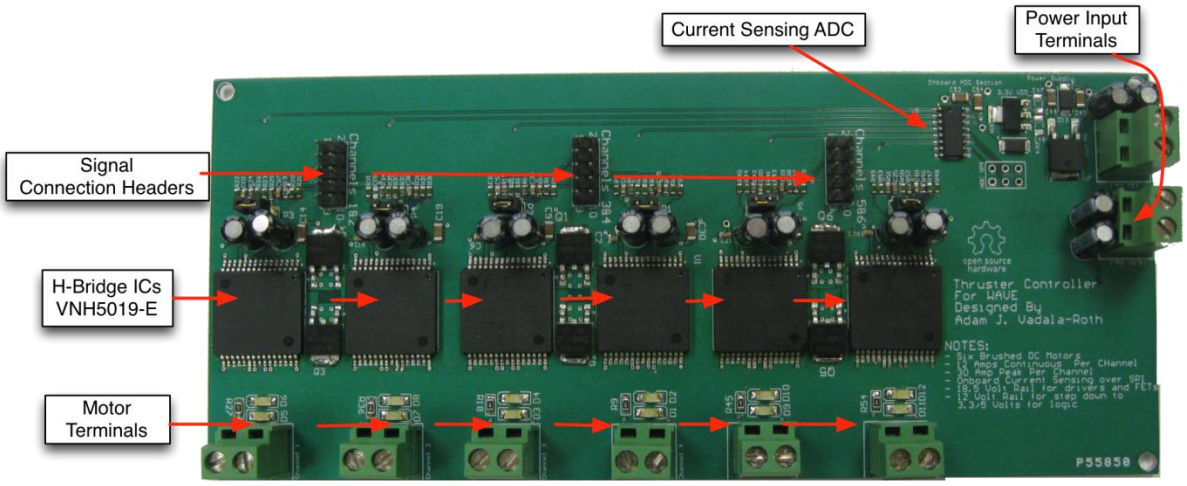


Figure 60: Thruster Controller Board

In the interest of building off the concept of modularity and rapid expandability of the AHD's Arduino shield compatibility high current Arduino shields were researched. On the market there were three high current DC motor control shields that were capable of controlling the chosen thrusters at their high current draw requirements; the Monster Moto Shield from Sparkfun.com, the Pololu Dual VNH5019 shield, and he Robot Power MegaMoto shield. While each of these shields were capable of handling the large continuous current draw of WAVE's thrusters they all had shortcomings that made them unfit for the task. The Monster Moto Shield from sparkfun.com while being capable of handling the current draw of the motors, could not handle the 18.5v that the thrusters were intended to be running at. The Robot Power MegMoto shield was only capable of driving one motor with direction control or two motors in one direction. Since WAVE uses a holonomic drive, the MegaMoto could not be used. That left the Pololu VNH5019 shield, which had the best specifications of them all, capable of driving two motors in any directions with 12A continuous current draw at voltage ranges from 5.5v to 24v. The Pololu shield had the correct specifications to drive WAVE's thrusters well, at least one pair of WAVE's three pairs of thrusters. The initial plan was to stack three of the Pololu shields on top of each

other on a single AHD for closed loop control of all movement in any direction but this plan had faults that prevented it from being executed.

The Pololu shield, while being perfect for the task, suffered from the following faults:

- Stacking the shields would be unsafe due to the large amount of heat dissipated by the VNH5019s on each shield,
- The pin out of the shield did not allow more than two shields to be stacked, and
- The current sensing signal from the VNH5019 was analog which would cause problems with sensing accuracy due to signal degradation.

While the Pololu shield had its faults and could not be used it had a lot of great features, but those features stemmed from the use of the ST Microelectronics VNH5019 H-bridge integrated circuit. The VNH5019 H-bridge had everything needed imaginable to drive a high current brushed DC motor precisely and safely (current sensing built in) so the decision was made to design a custom printed circuit board based around the ST Microelectronics VNH5019 H-bridge IC to drive all six of WAVE's thrusters.

The design of the custom thruster controller started from the source documents of the Pololu shield. Fortunately Pololu made the schematic for their shield available on their website; from the schematic and pictures of Pololu's shield PCB the electrical system designer was able to completely reverse engineer Pololu's shield and create WAVE's thruster controller. WAVE's thruster controller needed the ability to control six brushed DC motors but the Pololu shield circuit only covered two motors. The solution was to duplicate the circuit and PCB layout three times over to achieve six motor channels. First the Pololu shield schematic was re-laid out in Altium Designer then duplicated so that there were three schematic sheets, each sheet containing a pair of VNH5019s and their supporting circuitry. The next step was to unite these schematic documents under a master sheet in Altium to show

each pair of H-bridges as subsystems of the controller as a whole. Next the datasheet for the VNH5019 was consulted to make sure the Pololu circuitry followed the reference design guidelines. It was found that the Pololu design left out some of the recommended power filtering capacitors which were then added to the design. Lastly, a schematic sheet containing the power source connections and signal connections was added and linked into the design hierarchy of the Altium project. In essence the thruster controller is three of the Pololu shields on a single PCB with the control signals broken out to 2x5 pin headers for each pair of VNH5019s instead of Arduino headers. These three 2x5 pin headers would later be connected to the Navigation and Locomotion shield attached to the AHD responsible for the locomotion and navigation of WAVE.

As a final note the thruster controller was not simply a clone of the Pololu shield multiplied out three times – there were a few improvements added for better performance and safer operation. The first and foremost change was designing it as a four layer printed circuit board with the two outermost layers being for control and motor driver signals and the two internal layers being for ground and power. A four layer printed circuit board has the following advantages over a two layer board:

- Dedicated layer for power distribution
- Reduced signal interference with power in its own layer
- Dedicated layer for uniform ground plane
- A uniform ground plane between the two signal dielectrics allows for the shortest path of inductance to ground to be the same for all ground termination
- Provides the designer with the flexibility to adhere strictly to design guidelines more easily

The four layers allowed all the signals to be separated very nicely, the layer stack consisted of control signals and current sensing on the top layer, the second layer was the uniform ground plane, the third layer consisted of the power plane, and the bottom layer consisted of the motor control signal

copper pours. The chosen stack profile allowed the sensitive signals to be shielded from the power plane and motor signal pours leaving a very robust signal conscious design in the end. Another improvement over the stock Pololu design was better accommodation for the motor current draw. When designing a PCB to handle high currents, the traces of the high current signals need to be wide in order to make up for the lack of depth (PCBs are very flat as opposed to low gauge wire). Often times making traces wider can be problematic or annoying in a PCB design so instead of using traditional traces copper pours are used, copper pours are arbitrary polygons drawn in the place of traces, these polygons are filled in with solid copper. Copper pours tend to be wide, their wide profiles translate to a larger volume of solid copper which in turn translates to a higher current rating. When constructing the copper pours for the thruster controller as much space as possible on the bottom side of the PCB was taken up for the copper pours for the motor signals leaving them larger than the ones found on the Pololu shield. In addition to making the pours larger extra space on the power plane was cut away from the main power sections and repurposed as internal copper pours for each motor. The end result leaves WAVE's thruster controller having significantly more accommodation for high current motors than the stock Pololu shield. Lastly, the final improvement made to the stock design was better current sense data acquisition. The stock Pololu shield merely connected the analog signal output from each VNH5019 to a channel on the Arduino shield's analog header. In their situation that is perfectly fine since the path the signal has to travel is very short leaving the possibilities of signal degradation low. In the case of WAVE at the time of designing the thruster controller it wasn't clear how far the analog signals would have to travel to the AHD with the navigation and locomotion shield on it. The solution to the problem was to include an analog-to-digital converter on the thruster controller to digitize the signals as close to the source as possible allowing for minimal signal degradation. So in the final steps of the design a Microchip MCP3208 analog-to-digital converter was added to the thruster controller to digitize each of the six current sense signals and send the data back to the navigation and locomotion AHD over an SPI link. The

MCP3208 was chosen because it has an onboard multiplexer that allows for eight analog signals to be connected to discrete pins and the required software is trivial to write.

5.6.2.2 Navigation and Locomotion Shield

In order for WAVE to navigate through its environment from position-based navigation, a sensor giving position and velocity across all axis, control over all degrees of freedom, and the ability to configure all control loops as closed loops would be required. The Navigation and Locomotion shield, designed to run off a single AHD, achieves all of these goals and packs them neatly into a custom designed PCB. For measuring position and velocity WAVE has been outfitted with a Microstrain IMU/AHRS sensor that communicates over either RS232 serial or USB. For control over all degrees of freedom WAVE has its custom thruster controller.. The Navigation and Locomotion shield is responsible for uniting all these features into an integrated system for performing all navigation and motion control of WAVE in Cartesian space. The navigation and locomotion shield can be viewed in Figure 61.

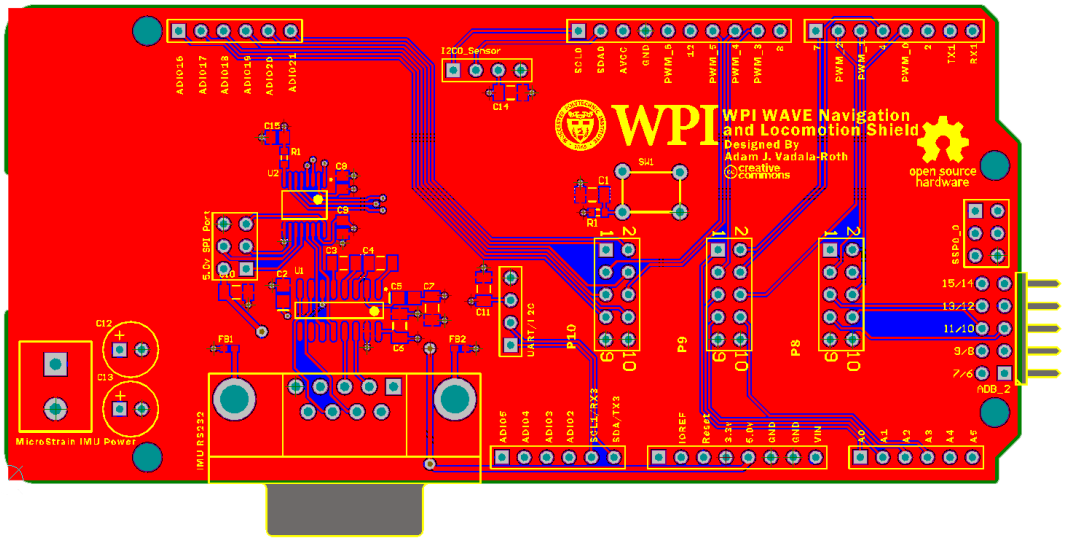


Figure 61: Navigation and Locomotion Shield

The design of the Navigation and Locomotion shield was met with very few design challenges, mainly because a lot of them were solved previously by the subsystems it unites together. The main design challenge with the shield was integrating sensor data into a closed loop control paradigm. The

IMU/AHRS being used could only communicate over RS232 serial or USB, the microcontroller onboard the AHD mainly communicates over lower level serial protocols. While it is fully capable of USB host functionality the software challenges to implement such integration would be far too challenging to complete in a reasonable enough time frame. The solution devised for integrating the IMU/AHRS was to include onboard conversion circuitry that will convert the RS232 signals down to 3.3v logic level UART signals that the AHD could easily work with. The serial conversion being the only design challenge, the shield design was relatively simple, consisting of a 2x5 pin header for each pair of H-bridges on the thruster controller, a 2x3 header for SPI coms, and lastly mapping all of the signals coming in on those headers to pins on the AHD.

The protocol conversion circuitry to step the RS232 down to 3.3v logic level UART consisted of two stages, shifting the RS232 down to UART with a shifting IC and converting the 5v logic level output into 3.3v logic level. The RS232 signals were shifted down using the popular Maxim Integrated MAX220 IC which is designed to shift standard RS232 from a DB9 connector into UART RX/TX at a logic level of 5v. The main problem with the MAX220 was that its output was 5v and the AHD was a 3.3v device, this problem was solved with the Texas Instruments TBX0106 bidirectional level shifter. With the help of the TBX0106 the 5v signals from the MAX220 were stepped down to 3.3v for the AHD.

Going through all the trouble of connecting the IMU/AHRS to the AHD is important because it enables closed loop control in real time. If the AHRS were to be connected to the Fit-PC over USB with the locomotion and navigation control loops running on the AHD the system would behave erratically and unpredictably because the control loops on the AHD would not be able to receive the data from the IMU/AHRS fast enough. The problem stems from the large amounts of latency introduced via USB communication in addition to the fact that the Fit-PC is not running a real time operating system nor is it equipped with custom written kernel extensions to hasten USB communication with the peripheral

device aboard WAVE. An easier solution was to develop circuitry onboard the navigation and locomotion shield to enable the IMU/AHRS to talk to the AHD over serial.

5.6.3 Sensor Board

In order to implement the following sensing modules: inertial measurement, temperature, humidity, flooding, and depth, the sensing modules were divided into two subunits: external and internal sensing modules. The internal sensing modules include the following sensors, which will be contained within the pressure vessel: inertial measurement unit, temperature sensors, humidity sensor, and flood sensors. The external sensing modules include any sensors that are contained outside of the pressure vessel, the only sensor that is designed to be mounted outside of the pressure vessel is the pressure sensor which is used to indicate depth. Due to time constraints this sensor was not able to be fully integrated into the robot but is designed to be connected to the system through one of the waterproof thru-hull connectors; the internal connections for the pressure sensor would have been integrated on the ballast board. A functional prototype of the sensor board can be viewed in Figure 62.

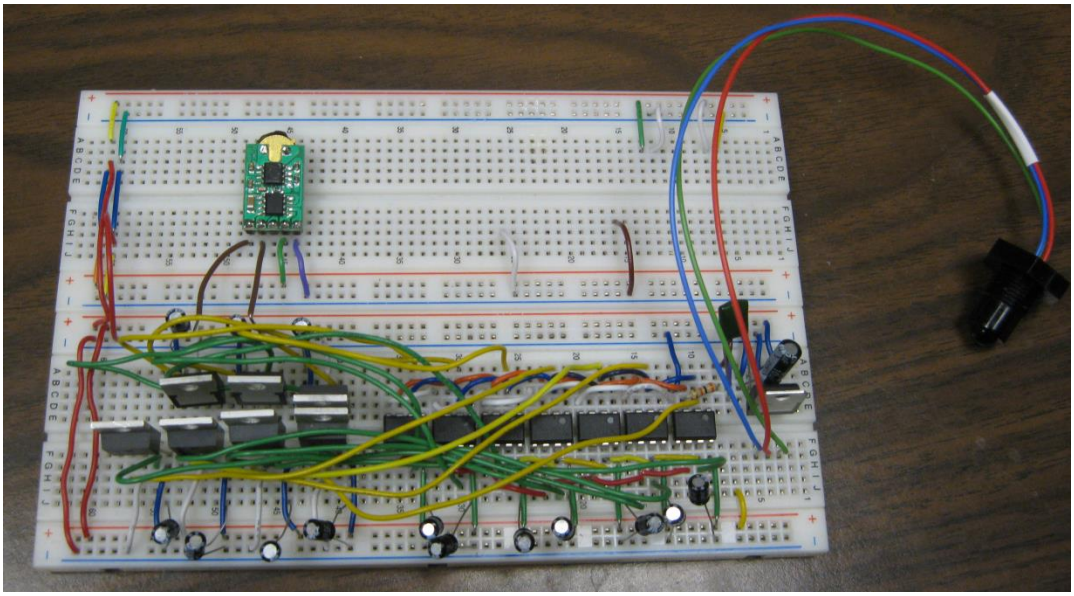


Figure 62: Internal Sensor Shield Prototype

The inertial measurement unit will be located on the navigation/motor board. The internal sensor shield will be housed within the pressure vessel and will contain the following sensors and support circuitry necessary to measure the following functionalities: temperature, humidity, and compartmental flooding. The internal sensor shield design is based upon the design of the Abstract Hardware Device. The internal sensor shield was an extension of the Abstract Hardware Device by the addition of connections for the temperature, humidity and flood sensors. The Microchip TC77 temperature sensors are designed to be mounted at seven different locations within the pressure vessel: three of the temperature sensors will be mounted on the batteries with straps, two sensors will be mounted to sense ambient temperature, and another temperature sensor will be mounted near the motor controller board to monitor the temperature of the motor driver chips. Due to size constraints the pressure vessel was only able to include two batteries and as such would only require the implementation of six temperature sensors. In order to connect to the shield itself each temperature sensor will be connected via jumper cables to the header connections on the shield. The headers for the temperature sensors were connected to I2C bus extenders to ensure the information from the temperature sensors could be communicated over a distance. This bus was part of the design because future teams may wish to redesign the electronics housing to a longer length. The design contains seven bus extenders, one for each temperature sensor. The humidity sensor is connected on the internal sensor shield and communicates over I2C as well. The information for the temperature and humidity sensors was read by connecting all of the sensors to the same SDA and SCL line. This information could then been be communicated via communication protocols that were compatible with I2C. The sensors are queried at a regular interval as specified by the high-level mission control. Two flood sensors are also connected to the internal sensor shield. The flood sensors are located on opposite ends of the pressure vessel. In order to connect to the shield itself the sensors were connected with jumper cables to header pins on the internal sensor shield. This sensor outputs a digital signal from the internal shield. Water was

detected in the system when the output was low, and no flooding was detected when the output was high.

The temperature and humidity sensors were designed to be powered through the voltage supply on the AHD. The temperature sensor would be powered with 5V and the humidity sensor would be powered with 3.3 V. As such a voltage divider was implemented to supply the humidity sensor with power. The flood sensor was also designed to be powered with the 5V supply on the AHD. With all of the sensors connected to the same voltage supply, protection circuitry was implemented in order to ensure that the system supplied the necessary power to all components. This protection circuitry consisted of two components: a voltage regulator and bypass capacitors. The voltage regulator was connected to the voltage supply in order to regulate that the voltage supplied was always 5.0 V. The bypass capacitors were located one each component to filter out any noise within the system that might be caused with the multitude of components and ensured that a steady 5.0 is supplied.

6 Software Design and Analysis

With such a complex system of controllers, sensors, and actuators, WAVE needed a robust software suite to coordinate its many parts. The robot must be capable of real time operations (such as PWM generation) and high-level mission planning. While a single computer could handle these tasks, it can be more effective to have multiple machines to handle the different tasks. Low level signal generation and sensor manipulation is best suited to an embedded environment, whereas mission planning requires high-level, memory and processor intensive computation. To coordinate all of these various functions, it was decided to distribute computation between several embedded microcontrollers and a centralized management computer. The robot may then be monitored via a remote interface, while it operates autonomously.

6.1 Distributed Processing

This software is responsible for a multitude of tasks such as gathering sensor data, controlling motion, path planning, inter-device communication, and communication with the poolside user interface. Keeping in line with WAVE's design goals of modularity and expandability, the processing has been distributed between a centralized controller, and each module's Abstract Hardware Device. As specified in section 5.2, these devices communicate over USB or Ethernet, and each has a designated functionality.

The various components of the robot's computational core require careful organization and planning. The communication paths must be clearly defined, and easily modified as necessary. As shown in Figure 63, the various tasks are distributed between the Mission Controller and the Abstract Hardware Devices. One critical aspect of this layered design is the path of the data used in control loops. For example, the control loop which would maintain WAVE's heading requires access to the accelerometer and gyroscope data. This control loop needs to run several times a second, using certain thrusters to control yaw. This data path must be contained within a single computing device, to avoid

saturating the USB controller with the transmission of time-sensitive heading data obtained on one controller that needs to be used in the PID control loop running on another controller. To avoid this constraint of resources, the modules were designed to include all relevant actuators and sensors on the same AHD.

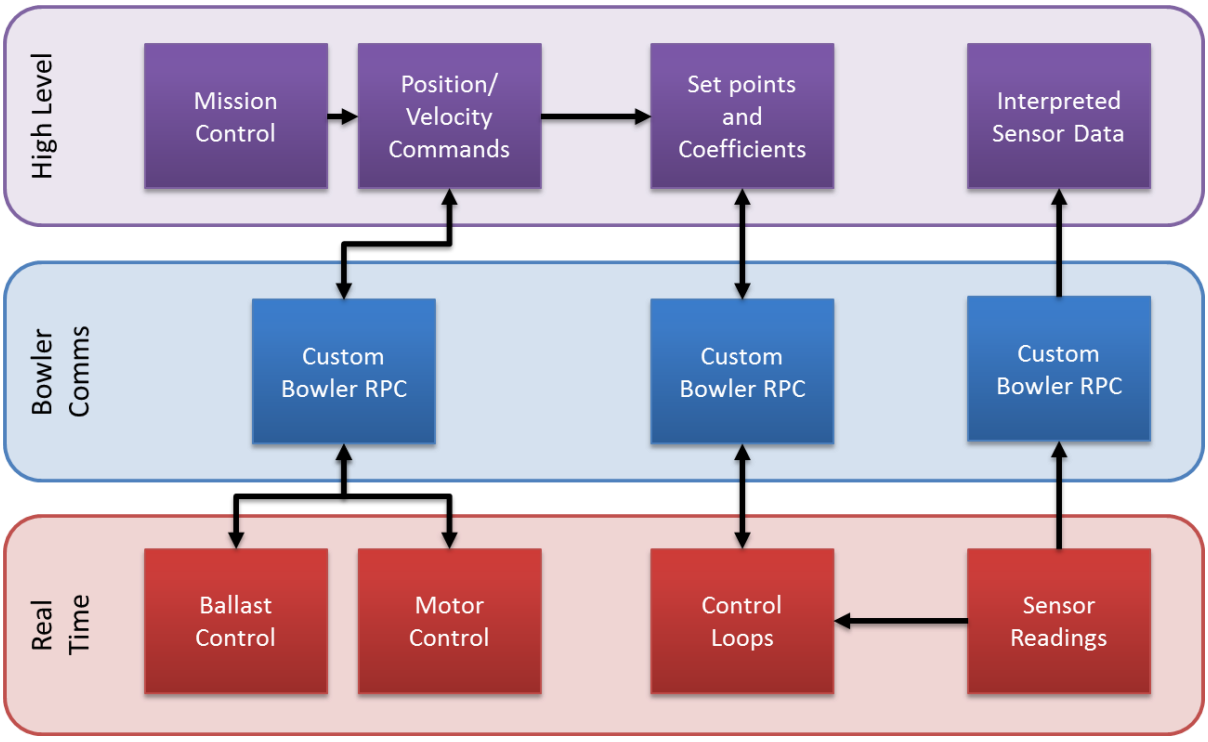


Figure 63: Critical Data Paths and Software Levels

The primary computer aboard the submarine was one of the only outstanding costs of the software team. This computer had to be capable of running all of the complex code driving the robot. Selecting the computer was done through the design matrix shown in Appendix C: fit-PC . This design matrix allowed the Software Team to compare various qualitative characteristics of the various computer solutions in a quantitative way. The team ultimately selected a fit-PC 3 because of its small form factor, low power consumption, and price.

The primary selection criteria for the main controller were small size, low power consumption, and enough processing power to enable even the most difficult processing tasks, such as computer

vision. The team considered two different styles of controller for use in the robot. The first alternative was a self-contained off-the-shelf computer, typically referred to as a Mini-PC. Mini-PCs have numerous advantages. They are typically the smallest, full-featured computers available. They are inherently power efficient, and contain their own power supply. However, Mini-PCs are not fully upgradable, as the processor is typically not removable. The second alternative was a custom computer, built by the software team, using commercially available parts. This would allow the team to custom select the components used inside the robot, and would leave room for future upgrades, including RAM, processor, and hard drive. However, this design is typically larger than a Mini-PC, and does not include a power supply.

Ultimately, the team elected to purchase a Mini-PC, specifically a fit-PC3, because it is a cost-effective, off the shelf solution. The decision matrix showing how the team chose a specific fit-PC model can be found in Appendix C: fit-PC Feature Comparison. The fit-PC requires only a 12V barrel jack for power, making it ideal for running inside WAVE. The team chose to install Ubuntu Server on the fit-PC, because of its extremely low overhead and vast support community.

6.2 Software and Environment Selection

The selection of software to complement the hardware on WAVE is critical to its success. A robust core library for communicating with embedded devices is crucial. By supporting the Neuron Robotics Software Development Kit (NRSDK) [56] with extra third party packages, the team was able to focus on the team's specific deliverables, tying several discrete components into a cohesive unit. Several key decisions streamlined the construction of the code-base, such as selection of Integrated Development Environment (IDE), cross-platform capabilities, and deployment methods. The team cut down development time significantly by developing software that could be run on Windows or Linux, using the Eclipse IDE, and deployed to the robot in the form of JAR (Java Archive) files.

6.2.1 Module Communication Framework

To orchestrate the robot's numerous subsystems, a centralized computer running high level object-oriented code was required. The primary task of this software was to parse and execute mission files. These mission files (detailed in section 6.4) contained various steps and waypoints to be executed in order by the Mission Controller. The team determined the Bowler Communications Suite from the Neuron Robotics Software Development Kit would best fit their needs, after examining numerous communications methods between the Fit-PC and module AHD boards.

In order to control the low level functions of the robot, the robot needs to implement a reliable communications framework between the main controller and the Abstract Hardware Devices. Robot Operating System (ROS), is widely-used in the robotics community, seemed like a natural choice for WAVE. ROS is installed atop an Ubuntu client, such as the fit-PC, and facilitates communication between various "nodes," each performing a specific task. However, upon closer inspection, ROS would require considerable configuration and code, including a device driver for the AHD. Also, this would require a first revision of the Abstract Hardware Device to be designed, built, and operational before any communication code could be tested by the CS team. The NRSDK, however, is designed around the Bowler Communications Protocol, a clearly defined RPC system for linking computers and microcontrollers. The NRSDK is written in Java, and has a robust modular framework, and is open source. Manufactured alongside the NRSDK is the DyIO, a robust microcontroller, capable of several complex functions out of the box. It features USB communications, and is controlled using the Bowler Communications Protocol. Ultimately, the software team decided to go with the solution from NR, because it met most of the framework requirements.

While developing the software suite, the software team was able to use the DyIO as a testing platform for their code, while the ECE team developed the AHD.

6.2.2 Software Development and Organization

Aside from the library selection outlined in the previous section, a critical decision was which environment to use while developing the software. The team's main criteria were to be able to develop and test outside the robot within the same environment, and to simplify deployment to WAVE's Fit-PC.

6.2.2.1 Integrated Development Environment and Code Repositories

In order to quickly and efficiently develop and test the code, the team elected to use the Eclipse Java IDE early on. Though memory intensive, this software suite allowed for the addition of several plugins to help guide the project. It was familiar to all members of the team, many of whom already had it installed on their machines. The "Subversive" plugin as added to connect Eclipse to a Subversion software revision repository which is hosted by Google code. For more details on the software repository and how to access WAVE's code see Appendix J: Code. Google's "Window Builder Pro" provided a graphical interface for GUI design, greatly simplifying the construction of the poolside interface. Eclipse comes prepackaged with Apache Ant, a simple script interface that can build and package Java code. This complements the JAR file deployment strategy (Outlined in Section 6.2.2.2).

6.2.2.2 Software Deployment

To upload code to the robot quickly and efficiently, the team chose JAR files to encapsulate the code. Java Archive files are the standard way to package several Java files for interpretation elsewhere. However, the JAR creation process can be a bit tedious; however, it is automated easily. As previously explained, the Eclipse platform includes the Apache Ant scripting library, which allows for simple cross-platform scripting for a variety of functions. These Ant scripts can be quite powerful, orchestrating several complex procedures. The buildfile organizes these tasks in the form of 'targets', one of which is shown here:

```
<target name="jar">

	<manifestclasspath property="lib.list" jarfile="${jarpath}">
		<classpath refid="build-classpath" />
	</manifestclasspath>
```

```
        <jar destfile="${jarpath}" basedir="."
        excludes="**" filesetmanifest="mergewithoutmain">
                <zipgroupfileset dir="${dir.lib-common}" includes="**.jar" />

                <fileset dir="${dir.build}" includes="**/*" excludes="META-INF/*.SF" />
                <fileset dir="." includes="${dir.media}/*" />
                <fileset dir="." includes="${dir.data}/*" />
                <exclude name="${dir.dist}/**"/>

                <manifest>
                        <attribute name="Main-Class" value="${main-class}" />
                        <attribute name="Class-Path" value="${lib.list}" />
                </manifest>
        </jar>

        <chmod file="${jarpath}" perm="+x" />
</target>
```

The green characters surrounded by `${ … }` represent variables specified at the beginning of the script file. This function will take all of the compiled `.class` files within `dir.build`, along with the supporting libraries in `dir.lib-common` and package them in a jar at `jarpath`. It includes all of the necessary information to run the JAR on another machine, including the class manifest, and the location of the main method. This file is then marked as executable and the function returns.

This target and many more were run hundreds of times during the development of WAVE, and are detailed in Appendix P: Ant Build File.

6.3 Closing the Feedback Loop

During the software team's design process, one critical constraint was discovered, which would considerably affect the performance of the robot. The sensors providing feedback to the PID loops must be present on the same Abstract Hardware Device as the actuators being controlled.

When running PID loops on the embedded controllers, frequent sensor input is required as feedback to the controller, modifying its output. Three possible data paths were considered, USB, point-to-point device link, or onboard. If the existing USB connection were used to move sensor data from one Abstract Hardware Device to another, there would be considerable delay. Round-trip time for any USB

device can easily be over 10ms, introducing unwanted latency into the PID loop. To counteract this latency, inter-device communication was considered. However, this would require significant modification to the existing Bowler Communications Protocol and supporting embedded code. This would also only reduce the latency, and not eliminate it. The third option, having the sensors and actuators on the same device, would nearly eliminate all delays, and allow the PID loops to run faster and more reliably. The AHDs would not require significant additional code, and only moderate hardware modifications. In order to mitigate the PID loop latency, the team elected to connect relevant sensors to the AHDs running the corresponding PID loops, the third option outlined above.

6.4 Mission Control

Mission control is one of the most important parts of WAVE's software architecture. Without a strong mission-control framework, WAVE would be unable to perform a complex series of tasks autonomously. Missions are controlled by the task manager and modeled using a mission model. A mission is given to WAVE via a user-constructed XML file; each mission is broken up into a series of task objects of various types representing the different actions WAVE can perform. Figure 64 shows a very simple mission file that blinks an LED.

```
<?xml version="1.0" encoding="UTF-8"?>
<Mission name="Test Mission">
    <Task type="Blink">
        <Device>BlinkTest</Device>
        <Port>1</Port>
        <Duration>1000</Duration>
        <Gap>1000</Gap>
        <Count>3</Count>
        <Inverted>False</Inverted>
    </Task>
    <Task type="Wait">
        <Duration>5000</Duration>
    </Task>
    <Task type="Blink">
        <Device>BlinkTest</Device>
        <Port>1</Port>
        <Duration>1000</Duration>
        <Gap>1000</Gap>
        <Count>3</Count>
        <Inverted>False</Inverted>
    </Task>
</Mission>
```

Figure 64: Sample Mission File

Line 1 is the XML declaration that gives the XML version and encoding. Line 2 is the root mission element that contains an attribute for the name of the mission. What follows are three task elements, two of type “Blink” starting at lines 3 and 14, and one of type “Wait” starting at line 11. The blink tasks have properties for the device to send commands to, the duration for the LED to remain on, along with the gap between blinks and how many times to blink (lines 4 – 9). The wait task simply includes the duration to wait for. So this mission first blinks the LED 3 times (for the LED is on for 1000 milliseconds each blink with 1000 milliseconds between blinks). It then waits for 5000 milliseconds and then runs an identical blink task to the previous one blinking the LED an additional three times. Additional sample Mission Files are included in Appendix O: Example Mission Files.

6.4.1 Mission Model and Task Manager

The Mission Model is part of the global robot model which will be discussed in detail in section 6.5, and represents the mission that WAVE is executing. The mission model takes a given XML file and parses it to determine what tasks it has to perform and the parameters of those tasks. The task manager then takes this mission model and takes the task list from it to begin execution of the mission.

The manager iterates through the list of tasks; for each task it first checks if the task is asynchronous and should be run in the background as detailed in 6.4.2. An example of this would be safety tasks such as monitoring temperature levels of the electronics. Asynchronous tasks, such as polling a sensor, or sending data periodically, are handled in parallel. These asynchronous tasks run for the duration of the robot’s operation or until a certain condition is met. When the `TaskManager` encounters an `AsynchronousTask` as it runs through the list, the `AsynchronousTask` is removed from the main mission stack and started in a separate thread. Synchronous tasks, however, cannot be executed in parallel. Synchronous tasks include navigating to a waypoint, rotating, or anything which must be executed in a particular order. These tasks “block” the task manager, which will not progress on to the

next task until they return. The main `runMisssion()` function of the `TaskManger` is included below.

```
public void runMission(){

	timer = new Timer((int) updateFrequency, this);
	timer.start();


	List<AbstractTask> taskList = mission.getObject().getTaskList();

	for(AbstractTask task : taskList){
		task.addChangeListener(this);
	}

	Log.V("TaskManager", "Starting a list of " + taskList.size() + " tasks.");

	for(AbstractTask task : taskList){
		if (task.isAsync()){
			AsynchronousTask asyncTask = ((AsynchronousTask)task);
			Log.I("TaskManager",
				"Starting Asynchronous task " + asyncTask.getName());
			mission.getObject().getAsynchronousTasks().add(asyncTask);
			asyncTask.start();
		}else{
			Log.I("TaskManager",
				"Starting Synchronous task " + task.getName());
			task.run();
		}
	}
}

@Override
public void stateChanged(ChangeEvent e) {
	hasChanged  = true;
}

@Override
public void actionPerformed(ActionEvent e) {
	if (hasChanged) {
		ObjectServer.send(
			RobotGlobalModels.getMissionModel().getObjectWithLock());
		RobotGlobalModels.getMissionModel().unlock();
		hasChanged = false;
		lastSentTime = Calendar.getInstance().getTimeInMillis();
	}
}
```

This method begins with initialization of a `Timer` object. As arguments, the Timer constructor takes a reference to an `ActionListener` object, and a period in milliseconds at which to call that

object's `actionPerformed()` method. `TaskManager` uses this callback to periodically update the connected GUIs of any changes to its `MissionModel`. The updating process and the communication requirements between robot and GUI are covered in further detail in section 6.9.1.

6.4.2 Tasks

Mission files must be structured in a standardized way in order to be correctly parsed by the custom XML parser in the Java software. The root element is called "Mission" with an attribute for a mission name which will display in the GUI and help differentiate between various missions. What follows are a listing of child elements which are each called a "Task." These require an attribute "type" which indicates what type of task element is, and this attribute is used by the parser to correctly take the information stored in each element and create the respective Java object for that task type. The sub-elements within each task element represent the various properties of the task objects, and vary based on the task. The XML parser will read each of these elements to create the fields given to the constructor for the given task, and if the tag is not found a null value will be put in that field of the task so it is advisable to always make sure to properly create all elements so incomplete tasks are not generated, potentially causing issues at runtime. What follows are several example mission files and descriptions of what the missions in them would do.

Tasks come in several types to handle different actions that WAVE can perform. All task classes extend the `AbstractTask` abstract class, which contains basic functionality and properties required for all tasks. This includes a `run()` method, setters and getters for task progress and status, as well as a name for the task. Implemented tasks fall into two categories, synchronous and asynchronous tasks. These types of tasks are differentiated by their effect on the robot. Synchronous tasks are the type which must be completed before proceeding on to the next task. These tasks include moving to a location before manipulating an object at that location. Asynchronous tasks are tasks that can be performed concurrently to the synchronous tasks, such as polling sensors or restarting a watchdog

timer. Classes that extend `AsynchronousTask` are run asynchronously, whereas classes that do not are performed in synchronous order. The task manager iterates through the interpreted list of tasks, invoking the `run()` method on each. The `run()` method within `AsynchronousTask` overrides the `run()` method it inherits from the `Runnable` interface. This causes this method call to return almost immediately, spawning a new thread to handle this task.

Asynchronous tasks that have been implemented are tasks for sensor polling and blinking LEDs. The LED blinking task shown earlier was part of the initial testing of the software using DyIOs, and could also be used for debugging. The DyIO-based testing will be further detailed in section 8.3.4. Once basic Asynchronous functionality was implemented, more complicated sensor polling was implemented, in particular an `AHRSPollingTask` for polling WAVE's IMU. This task simply polls the IMU at a specified period and then sends updated values to the GUI to update the Attitude Indicator which is detailed in section 6.9.3.3.

Synchronous tasks that have been implemented include waiting and locomotion tasks, tasks for driving a testing basebot, and other debugging tasks. `LocomotionTask` sets up the framework of how to communicate with WAVE's motors in order to travel in the pool. As WAVE did not reach full system integration by the end of this project this task could not be fully implemented. There are two different wait tasks implemented, a generic `WaitTask`, and a `WaitForGUITask`. `WaitTask` simply waits for a given time in milliseconds, while `WaitForGUITask` waits until a GUI connects so that if debugging has to happen WAVE does not start running before a user can view all debugging information in the GUI. Without a fully assembled robot a smaller basebot was acquired in order to test the functionality of the software without being able to test on WAVE itself. Modified locomotion tasks were created to send commands to this basebot to drive its motors. Additional details on the basebot testing can be found in section 8.3. Debugging tasks were written to send echo messages to the GUI and to

blink LEDs in order to better test communications between the fit-PC, the GUI, and the embedded systems.

6.5 Robot Models

Measurements of its own state and that of its environment are central to the operation of WAVE. There are a number of detailed Model classes, which describe the various facets of the robot. These models contain a number of sub-models, allowing the user to easily encapsulate a particular system into a set of models. For example, the `RobotModel` contains two `TransformMatrix` models, discussed in the following Utility Functions section, representing the robot's current position, and the robot's desired position. These models are used to drive the robot's motion planning, and are periodically sent to the GUI for display. The `MissionModel` discussed in section 6.4.1 is another such model, which is included in the main `RobotModel`.

6.6 Libraries and Utility Functions

There are numerous classes created to serve a single particular purpose, which should be accessible from anywhere in the code-base. These types of files are frequently referred to as "utility" functions and classes. Several core utility classes have been outlined below.

6.6.1 Transform Matrix

As learned in RBE 3001, Transform Matrices provide a simple and elegant way to reference coordinates in 3D space, easily performing coordinate frame transformations without the need to implement these complex mathematical operations every time. The `TransformMatrix` class provides a simple implementation of the classic transformation matrix. These objects are handy for encoding the roll, pitch, and yaw angles, and allowing them to be chained together, or easily compared to one another. The matrix equations used to populate the `TransformMatrix` object are shown below, followed by the corresponding code snippet.

$$\begin{bmatrix} \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)\cos(\gamma) - \sin(\alpha)\sin(\gamma) & x_t \\ \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta)\sin(\gamma) - cos(\alpha)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\gamma) - \cos(\alpha)\sin(\gamma) & y_t \\ -\sin(\beta) & \cos(\beta) & \cos(\beta)\cos(\gamma) & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
private void buildMatrix(double roll, double pitch, double yaw,
      double x, double y, double z) {
      // Roll  = Gamma
      // Pitch = Beta
      // Yaw   = Alpha
      double ca = Math.cos(yaw);
      double cb = Math.cos(pitch);
      double cg = Math.cos(roll);
      double sa = Math.sin(yaw);
      double sb = Math.sin(pitch);
      double sg = Math.sin(roll);

      this.set(new SimpleMatrix(new double[][]
            {{ca*cb,     ((ca*sb*sg) - (sa*cg)), ((ca*sb*cg) - (sa*sg)), x},
            {sa*cb,      ((sa*sb*sg) + (ca*cg)), ((sa*sb*cg) - (ca*sg)), y},
            {-1*sb,      cb*sg,           cb*cg,                         z},
            {0,          0,                  0,                          1}}));
}
```

6.6.2 Logs

Monitoring the dozens of concurrent events occurring on WAVE can be a complex task. Printing directly to the console results in messages mixed together and inconsistently formatted, when printing is done in multiple threads. To standardize these messages, and organize them chronologically, WAVE implemented a logging system, with parameterized entries. Each log entry contained a number of fields, outlined here:

- Time – An object of type `java.util.Date` that stores the date that the entry was created. This field is automatically populated by the constructor, and accessible through a getter method.
- LogLevel – A custom enumerator object which stores one of the Log's 5 logging levels. These levels are prioritized in ascending order: Verbose, Debug, Info, Warning, and Error. This allows these messages to be filtered by the Poolside Interface but still recorded by WAVE.
- Topic – A string representing the topic which this entry pertains to. These topics are passed to the constructor, allowing the user to "group" entries pertaining to a particular topic.
- Thread ID – A string field containing the name of the thread which generated the message. In a multithreaded environment, these strings will help to identify the exact source of the message.
- Message – The textual message wished to be logged by the logging system.

The logging base system contains a simple queue for new messages. Upon startup, the Log will spawn a new thread, which simply waits for new entries to be added to the queue. The logging thread code is actually a single line, shown here:

```
while(true) Log.notifyListeners(Log.pendingEntries.take());
```

The log sends these entries to all registered listeners, each implementing the `LogListener` interface. These listeners register with the log, and are notified of each new entry the log receives. This Observer Pattern allows the log to scale easily to several observers, while the separate thread and message queue allow the Log publishing functions to return extremely quickly. Some key `LogListeners` included in the `Log` package are the `LogArchive`, `ConsoleLog`, and `FileLog`.

A single `LogArchive` object is registered with the `Log`, when the robot starts up. This archive stores a copy of each entry generated, for polling by GUIs. This allows the individual entries to be broadcast at generation, as opposed to sending the ever growing log archive. When objects like the GUIs require access to the previous log entries, they simply send a `Command` object (outlined in section 6.9.2) requesting the Log `Archive`. The `DefaultCommandHandler` responds by sending the `LogArchive` object stored in the Log.

The `ConsoleLog` simply echoes every log entry to the standard Java output stream, which may be mixed with other standard console output. The `FileLog` functions similarly, printing each `LogEntry` to a new line in a plaintext file, specified in the `FileLog`'s constructor.

6.7 Communications

Communications are an important aspect of WAVE's operations. Different robot components have to communicate with and exchange data between each other. Not only that, but the poolside interface has to be able to "talk" with the Fit-PC and all AHDs. This section will detail the various methods of communication between components.

6.7.1 Serial Communications

Serial communications are central to the AUV's operation. Reliable communication between the Fit-PC and any number of serial devices is critical to the robot's modularity and expandability. Included in the NRSDK is a library called NRSerial. This library allows for simple, cross-platform communication between any attached serial devices. The baud rate and target device are selected in the serial connection constructor, and is used in communicating with the AHDs, the DyIO, and the AHRS for testing.

6.7.2 AHRS Serial Communications

The Microstrain 3DM-GX3-25 9-axis IMU (3 gyros, 3 accelerometers, 3 magnetometers) provides two modes of serial communication. The first is their Microstrain Inertial Product (MIP) Data Communications Protocol. This protocol has two options for data transmission, the first and more complex of the two uses multi-byte command and response packets, providing greater flexibility, but a more complex implementation on the receiving side. The team opted to use the Single Byte Communication Protocol, where a single byte is sent to the AHRS, initiating a multi-byte response. A few key methods from the USB AHRS testing class are shown below.

```
private synchronized static ByteBuffer getBytes(int bytes)
	throws IOException {
	buffer = ByteBuffer.allocate(bytes);
	for (int i = 0; i < bytes; i++) {
		buffer.put((byte) dataInStream.read());
	}
	buffer.position(0);
	return buffer;
}
```

The `getBytes()` method returns a number of bytes from the serial data input stream. "dataInStream" is a static variable, initialized by the `AHRSManager.connect(String port)` method. This function blocks until it has received the number of bytes specified by the "bytes" parameter. These bytes are assembled into an object of type `ByteBuffer` and returned.

```
private static float getFloatFromStream() throws IOException {
	return getBytes(4).getFloat();
}
```

This `getFloatFromStream()` function simply wraps a call to the above getBytes() function. This function reads in the next four bytes from the input stream, and parses them as a 32 bit IEEE-754 floating point integer. The endianness of the AHRS data, specified in its specification sheet, matches the big endianness of Java, allowing these values to be assembled easily.

```
public static TransformMatrix getAttitudeMatrix() throws IOException {
	sendCommand(getAttitudeCommandByte);

	TransformMatrix result = new TransformMatrix();
	result.setRoll((double) getFloatFromStream());
	result.setPitch((double) getFloatFromStream());
	result.setYaw((double) getFloatFromStream());
	getBytes(6);
	buffer.clear();
	return result;
}
```

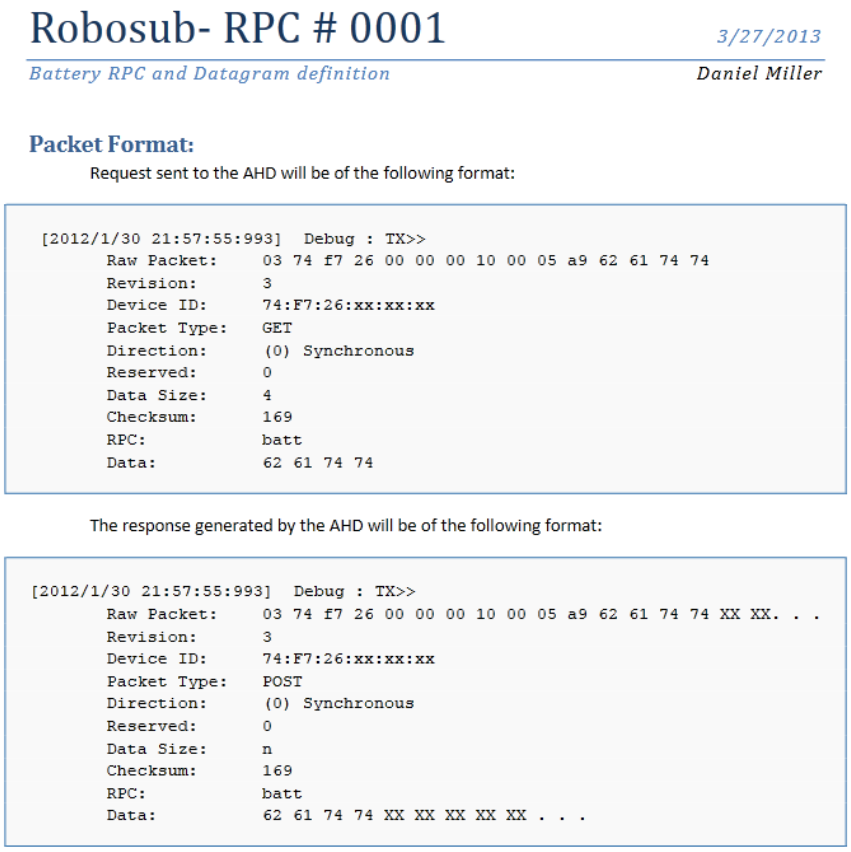
This static method, utilized by several external classes, sends a command to the AHRS, and parses all of its returned data. It adds it to a new `TransformMatrix` object, and returns it. This simple method quickly and accurately returns the full orientation matrix, relative to the fixed earth

coordinate frame, oriented northward. Position reckoning uses this method very frequently, when run over USB. Rather than USB communications, the AHRS was designed to be polled by the AHD. This would allow for much faster interrupt-driven sensor polling, which would result in much more accurate position reckoning.

6.7.3 Remote Procedure Calls

RPCs, or Remote Procedure Calls, lie at the core of the Bowler Communication Protocol. In the NRSDK, these calls take the form of serial packets, known as Bowler Datagrams. Contained within are several header fields, which define several parameters of the RPC transaction to be made. Some key header elements include the source and destination MAC addresses used on the USB network, and the RPC code field, which specifies the procedure that this packet represents. These RPC codes are four bytes long, and typically correspond to an abbreviated word, represented in ASCII. For example, the WAVE's battery health status RPC uses the code "batt". The payload of the Bowler Datagram is dependent on the RPC specified, and can vary between 0 and 251 bytes.

The WAVE team outlined several custom RPC calls, and detailed their contents and usage. An example packet layout is shown here, and the full details for all RPC calls can be found in Appendix N: RPCs.



Robosub- RPC # 0001 3/27/2013

Battery RPC and Datagram definition *Daniel Miller*

Packet Format:

Request sent to the AHD will be of the following format:

```
[2012/1/30 21:57:55:993]  Debug : TX>>
       Raw Packet:     03 74 f7 26 00 00 00 10 00 05 a9 62 61 74 74
       Revision:       3
       Device ID:      74:F7:26:xx:xx:xx
       Packet Type:    GET
       Direction:      (0) Synchronous
       Reserved:       0
       Data Size:      4
       Checksum:       169
       RPC:            batt
       Data:           62 61 74 74
```

The response generated by the AHD will be of the following format:

```
[2012/1/30 21:57:55:993]  Debug : TX>>
       Raw Packet:     03 74 f7 26 00 00 00 10 00 05 a9 62 61 74 74 XX XX. . .
       Revision:       3
       Device ID:      74:F7:26:xx:xx:xx
       Packet Type:    POST
       Direction:      (0) Synchronous
       Reserved:       0
       Data Size:      n
       Checksum:       169
       RPC:            batt
       Data:           62 61 74 74 XX XX XX XX XX . . .
```

Figure 65: Example RPC Specification

6.7.4 Device Factory and Configuration

In order for the system to know the serial number and path of each AHD, a Device Factory function was utilized. This function utilizes a simple text file in order to configure the various computing-related devices that have been incorporated into the submarine. This file contains the unique identifier tag of each device, followed by the Windows and Linux hardware path of the device. Depending on the operating system, the appropriate path is assigned to each device. The reason both paths are present in the file is two-fold: first, because the serial number of each device is contained within the Linux path, and second, testing and deployment used different operating systems – tests were performed using Windows and those tests needed the path in Windows, while the deployed submarine runs on Linux.

This file will be used by the Device Factory. It will store all the information about the devices in two tables. One table will map devices to their serial numbers, and the other will map devices to their hardware path. These tables can be accessed through the Factory in order to retrieve serial numbers and/or hardware paths of specific devices.

6.8 Human-Robot Interaction

During WAVE's underwater operations, a method for real-time system monitoring and debugging was required. For this purpose, a poolside graphical user interface was developed. The goal of this GUI is to provide a graphical depiction of the status of the robot, and provide feedback that assists with debugging. This interface must display mission-critical system information, such as battery life and memory usage, as well as output from the sensors, such as temperature and humidity inside the robot, relative position, path, heading and velocity. Additionally, the interface must allow remote activation of safety features, such as emergency stop, reboot or surface.

As in any user interface, WAVE's GUI must be easily understood, and able to easily and reliably convey information to the observer. The two main options for the robot's interface were either to provide a strictly textual interface accessed by SSH or to create a graphical user environment. A plain console has many advantages, such as extremely low overhead and a simple implementation. However, a graphical user interface allows us to efficiently represent complex data. However, the rendering of a graphical interface can impart more overhead on the main system, and requires significantly more time to develop. To counteract these additional computational requirements, the team designed a separate GUI program, run on the user's machine, which connects to the submarine over the local network. The submarine will then require only periodic updates to be sent to all connected machines, and not require a graphical environment to be run on the fit-PC.

Additionally, WAVE needed a way to connect to the GUI when submerged during testing. This was necessary in order to get real-time data, as well as a graphical indication of how the robot was performing in the environment it was built for. For this purpose, it was deemed appropriate to utilize a communications tether. WAVE and the poolside interface would connect via a waterproofed Ethernet cable. This connection would allow real-time upload/modification of mission files, as well as code changes on the fit-PC. This tether would need to be of sufficient length, so even at the bottom of the pool, the robot could stay connected.

6.9 Poolside Interface

In this section, a more detailed explanation of the GUI's features will be provided. These features include: details about the methods of communication between the robot and the interface, command packets utilized by the robot, GUI components and their functionality, along with screenshots of said components.

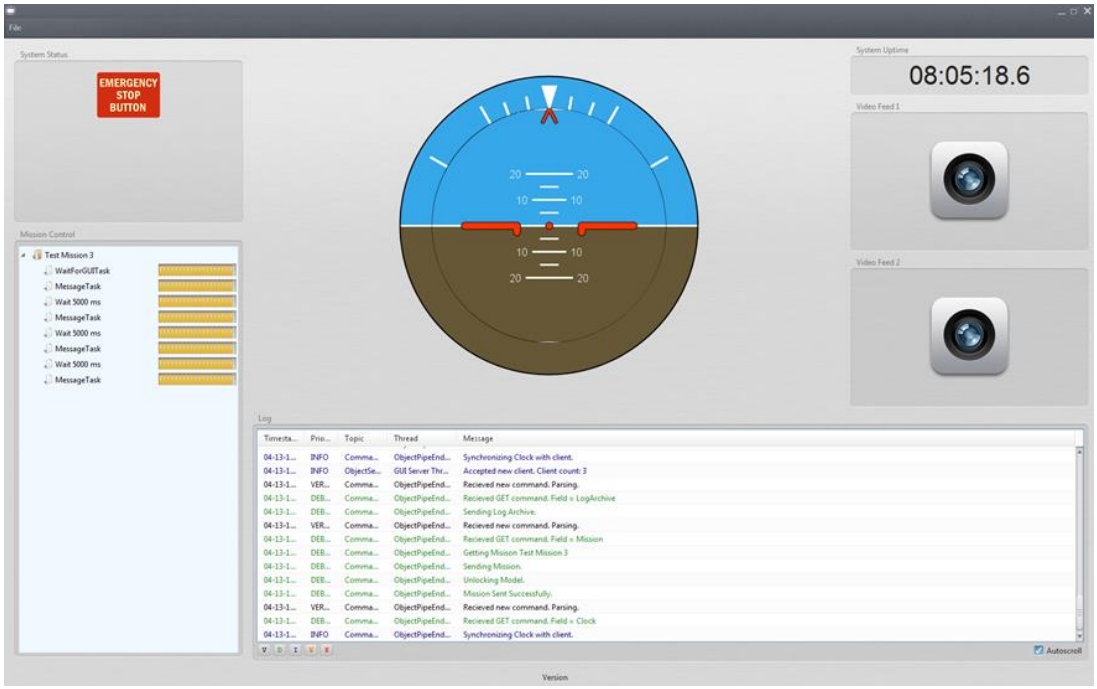


Figure 66: GUI Screenshot

6.9.1 Communications with WAVE

Before the Poolside Interface (commonly referred to as the GUI and shown in Figure 66) can display any data, a bidirectional connection must be established. As with all other aspects of WAVE, both the GUI and its connection to the robot must be modular, keeping in line with the original design goals.

6.9.1.1 Model/View/Controller Pattern

The Model View Controller (MVC) design pattern allows robust construction of user applications, separating data objects from its visual representations and the methods that manipulate the model. This pattern scales well to include dozens, if not hundreds of models, views, and controllers.

On WAVE, however, the views exist only on the GUI interfaces, and the controllers are really the environment in which the robot operates. WAVE's various sensors detect changes, both within and exterior to the robot, and update the various models accordingly. These updated models are then sent to all of the connected GUIs, where they will be informed of this change, so that the views may update.

In order to bridge this gap, the WAVE team developed a modified MVC pattern, to include a network socket through which models may be transmitted to the GUI. A block diagram of the implemented MVC pattern is shown in Figure 67.

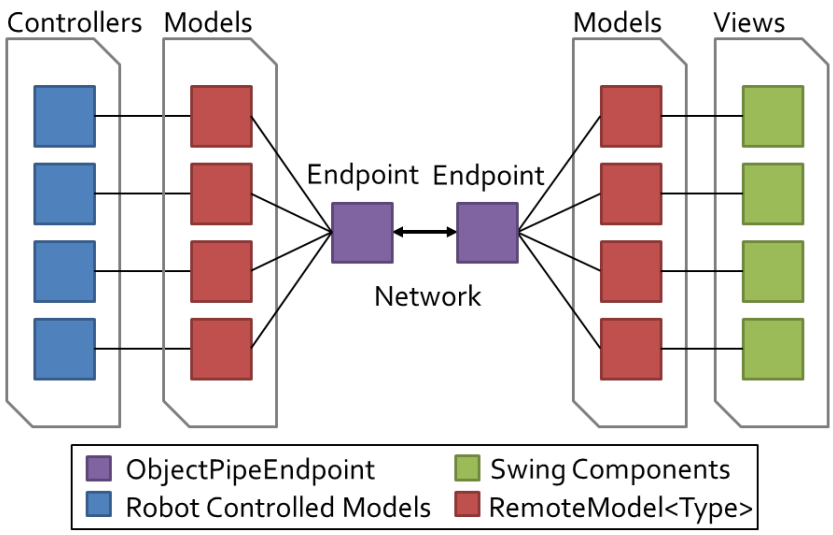


Figure 67: Model View Controller Diagram

The endpoints are responsible for serializing the given models, transmitting them across the network, and de-serializing the objects at the receiving end. This process, and all the classes involved are detailed in the following section.

6.9.1.2 The ObjectServer, ObjectPipeEndpoints, and Serialization

The process of accepting GUI connections on the WAVE platform begins on the Fit-PC with the `ObjectServer`. The `ObjectServer` combines several static methods and fields, and is initiated with the start of the robot. Starting the `ObjectServer` will spawn a new thread, which simply spins in a loop, accepting clients as they connect to the server's listening port (default port number is 4000). This loop, upon detecting a new connection, spawns a new `ObjectPipeEndpoint` to handle communications, and passes to its constructor the newly generated socket. This new `ObjectPipeEndpoint` is then registered with the `ObjectServer`, which adds it to the server's list of connected GUIs. The new connection is added to the Log, and the cycle repeats. This exact code loop can be seen here. This loop runs quite quickly, allowing multiple users to connect in quick succession.

```
@Override
public void run(){
	while (running) {
		try {
			// Block while waiting for a new connection
			ObjectPipeEndpoint client =
				new ObjectPipeEndpoint(server.accept());
			// Register this new client with the ObjectServer
			register(client);
			// Log this connection
			Log.I(ObjectServer.class.getSimpleName(),
				"Accepted new client. Client count: " +
				openSockets.size());
			// Repeat
		} catch (IOException e1) {
			running = false;
		}
	}
}
```

This same code has been translated into a network timing diagram, which helps to illustrate the order of events, and the components which invoke them. As shown on the right of the diagram in Figure 68, the connection is initiated by the GUI, and accepted by the `ObjectServer`. Upon accepting this GUI, the `ObjectServer` passes responsibility to a new `ObjectPipeEndpoint`, which handles all further communication.

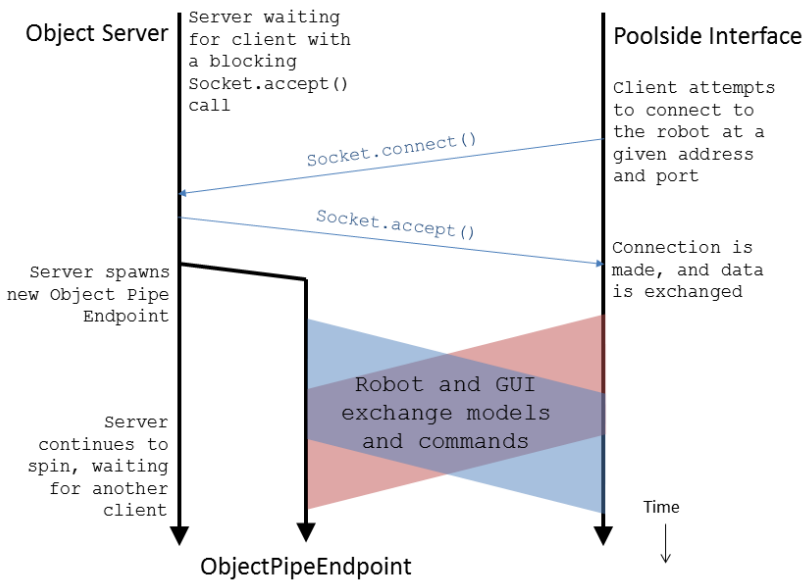


Figure 68: Network timing diagram for GUI connections

As mentioned previously, WAVE is designed to support multiple GUI connections. This is where the `ObjectServer.register()` method comes in. Every `ObjectPipeEndpoint` that is spawned by the `ObjectServer` is also added to a list of connected GUIs. Every time a model is updated on the robot, the program calls the `ObjectServer.send(Serializable obj)` method, which flips through the list of connected GUIs, and sends along the new model (or any object, for that matter.) The code for this send method is included here. This method iterates over the entire list of open sockets, invoking their `send(Serializable)` method individually. In the event that the given endpoint is no longer active, it is removed from the list of open connections.

```
public synchronized static void send(Serializable obj) {
	for (int i = 0; i < openSockets.size(); i++) {
		ObjectPipeEndpoint client = openSockets.get(i);
		if(client.isRunning()){
			client.send(obj);
		} else {
			openSockets.remove(client);
			Log.I("ClientConnection",
				"A client disconnected. Client count: " +
				openSockets.size());
		}
	}
}
```

The ObjectPipeEndpoint objects encapsulate both the input and output streams for objects, providing a simple interface for sending and receiving. The constructor takes in a socket as a parameter, which is assumed to be already connected. The input and output streams of this socket object are passed as constructor parameters to the `ObjectPipeEndpoint`'s `ObjectInputStream` and `ObjectOutputStream` fields, respectively. The endpoint simply serializes objects for sending, and feeds them through the ObjectOutputStream. Upon creation, the `ObjectPipeEndpoint` also spawns a thread, used for reading input from the `ObjectInputStream`. As new objects are received, they are de-serialized, and distributed to all classes which are currently observing this endpoint for a particular type of object. This special form of the Observer/Observable pattern was created specially by the WAVE team, and is referred to as the "Type Observable" pattern.

6.9.1.3 The Type Observable pattern

To facilitate the distribution of new objects from an ObjectPipeEndpoint, the team designed a new class, `TypeObservable`. This class, similar to the `Observer` class, distributes objects to registered objects via their inherited `notify(Object obj)` method. Just as `TypeObservable` extends the standard `Observable` object, it is observed by `TypeObserver objects`, each implementing the `Observer` interface.

This new Type Observable pattern nearly mirrors the Observer pattern directly, with one key modification. When a TypeObserver registers with a TypeObservable object, they include an example object of the type of object they would like to observe for. Rather than storing all registered objects in a simple `Vector`, as does the `Observable` class, the `TypeObservable` takes the class name of the example object and uses this as the Key to index into a `HashTable`. The values of this hash table are vectors, containing references to all of the objects which have registered for the type represented by the generated key. The `addObserver(TypeObserver o)` method is included here, illustrating how observers are added to the hash table.

```
/**
 * This method adds a new observer to this TypeObservable object.
 * @param o TypeObserver The object to be notified upon the receipt of a new object
 * @param key String This string will represent the
 */
public synchronized <T> void addObserver(TypeObserver o, String key) {
	if (o == null)
		throw new NullPointerException();
	if(!observers.containsKey(key))
		observers.put(key, new Vector<TypeObserver>());
	if (!observers.get(key).contains(o)) {
		observers.get(key).addElement(o);
	}
}
/**
 * Generates the string representation of the given object
 */
private String genKey(Object arg) {
	return arg.getClass().getName();
}
```

The second function `genKey(Object arg)` is a simple wrapper for the class string retrieval method. The `notifyObservers(Object arg)` follows, showing just how the `TypeObserver` utilizes the string hash map. When a new Object is presented to the `TypeObservable`, its class's string equivalent is generated by the `genKey()` method. This string is then used as the key, indexing into the hash map, retrieving the corresponding list. This list contains all of the observing objects for the type of the given object. This object is then copied to a simple array which is then iterated over,

notifying all of the observing classes. If no classes are currently observing for the given type, the array list will return null, which is converted to an empty vector. This empty vector is then iterated over, returning immediately, while notifying no objects.

```
public void notifyObservers(Object arg) {
	/*
	 * a temporary array buffer, used as a snapshot of the state of current
	 * Observers.
	 */
	Object[] arrLocal;

	//Synchronize this section, preventing concurrent modifications when adding a
	new observer
	synchronized (this) {
		if (!changed)
			return;

		Vector<TypeObserver> v = observers.get(genKey(arg));
		if(v != null) {
			arrLocal = v.toArray();
		} else {
			arrLocal = new Object[0];
		}
		clearChanged();
	}

	// Loop through the snapshot array, calling the notify methods of all
	registered objects
	for (int i = arrLocal.length - 1; i >= 0; i--)
		((TypeObserver) arrLocal[i]).update(this, arg);
}
```

The `TypeObserver` pattern enables the `ObjectPipeEndpoint` to distribute new objects as they are sent to the various poolside interfaces. While these endpoints are bi-directional, it does not enable the GUIs to command the robot itself in any way.

6.9.2 Command Objects

`Command` Objects enable the GUIs to send commands to the robot, which alter its behavior, or indicate emergency states. The primary use of these objects is to retrieve large pieces of data from the robot. For example, rather than constantly sending out the entire log each time a new entry is added, WAVE simply broadcasts the latest `LogEntry` to all connected GUIs. In order to maintain a list of all of the log

entries since startup displayed on the GUI, each GUI will send to the robot a `Command` object with "archive" as the command field. This field is parsed by the `DefaultCommandHandler` aboard WAVE, which is then responded to directly. Other commands implemented aboard this first iteration of WAVE include requesting the mission model, and initiating an emergency stop.

6.9.3 GUI Components

The system supports multiple GUI connections to the robot. A number of components have been developed to assist with robot controls, mission planning and debugging. These components are included in the poolside interface.

6.9.3.1 Emergency Stop Button

To prevent damage in case of a malfunction, an emergency stop button (shown in Figure 69) halts all current activities and makes the robot surface. This command has the highest priority and cannot be canceled and/or interrupted by another command.

Figure 69: Emergency Stop Button

6.9.3.2 Mission Control

Robot missions were made using a mission control tree/stack model shown in Figure 70. It lists all of the past, current and future tasks to be performed by the robot. This GUI component is populated from the stream of `MissionModel` objects sent by WAVE. As the `TaskManager` steps through these tasks, their progress is updated accordingly. The list is scrollable if not all tasks fit on screen.

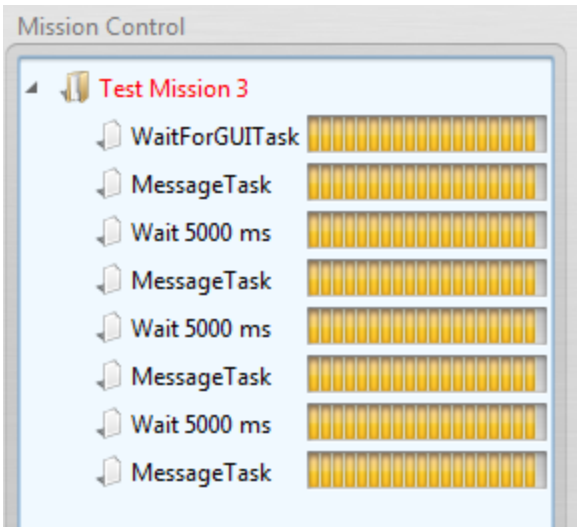


Figure 70: Mission tree showing an example mission, with all tasks completed

6.9.3.3 Attitude Indicator

Proper robot function requires various measurements to be taken. An attitude indicator (shown in Figure 71) shows the orientation of the robot, as well as roll, pitch and yaw. These measurements are taken from the IMU and converted into coordinates to update the indicator.

Each of the indicator's components, such as the horizon and horizontal lines, are drawn on a separate image first, and then combined to form the attitude indicator, as seen below.

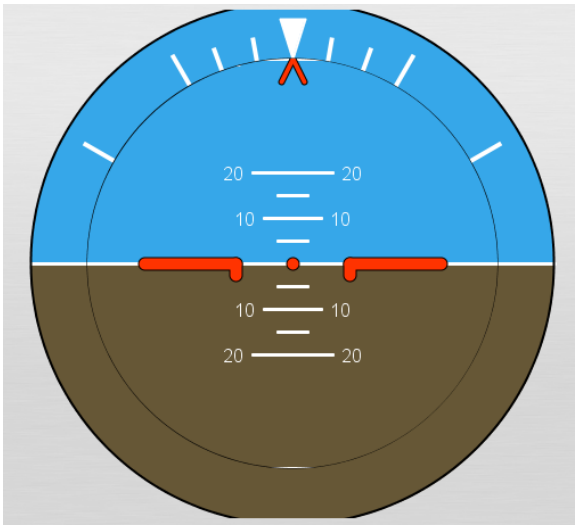


Figure 71: The attitude indicator

In addition, rather than redrawing the images on every update, they are transformed using an `AffineTransform` matrix, which sets the images' new orientation. First the center of the image is generated based on the image width and height:

```
double locationX = ringImage.getWidth() / 2;
double locationY = ringImage.getHeight() / 2;
```

Then, based on the location, the inner circle of the indicator is updated using the `AffineTransform` matrix:

```
AffineTransform rollTransform = AffineTransform.getRotateInstance(-1*roll,
locationX, locationY);
AffineTransformOp rollImage = new AffineTransformOp(rollTransform,
AffineTransformOp.TYPE_BICUBIC);
return rollImage.filter(ringImage, null);
```

6.9.3.4 Log

Messages, alerts and errors are displayed in a log. The log output displays robot and GUI specific messages, errors, warnings and exceptions. The log supports different filtering options, based on the type of message.

Log

Timestamp	Priority	Topic	Thread	Message
04-15-13 10:32:10.945	VERBOSE	ObjectServer	main	Server port changed to 4000
04-15-13 10:32:10.950	INFO	ObjectServer	main	GUI Server started on port 4000
04-15-13 10:32:11.183	VERBOSE	RunMode	main	The system is in DEPLOYMENT mode.
04-15-13 10:32:11.183	DEBUG	TaskManager	main	Starting Task Manager
04-15-13 10:32:11.191	VERBOSE	TaskManager	main	Starting a list of 8 tasks.
04-15-13 10:32:11.191	VERBOSE	TaskManager	main	Starting Synchronus task WaitForGUITask

Figure 72: The log having several messages generated upon startup

6.9.3.5 Timer

To help with mission control, the GUI contains a timer (shown in Figure 73) to keep track of mission uptime (each newly connected GUI will start from however long the robot has been on). Placeholder areas for two cameras showing real-time feeds have been provided.

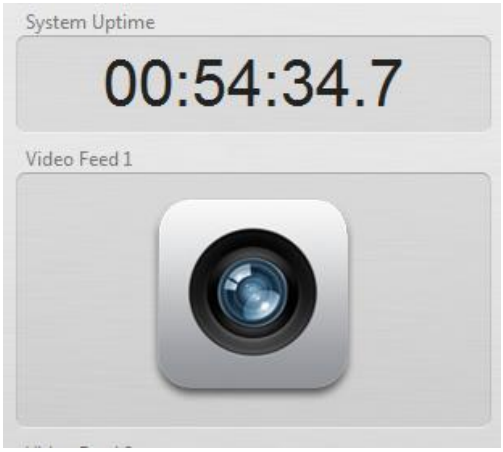


Figure 73: System uptime, along with a placeholder video area

7 System Integration

Integration of WAVE started upon the completion of the design and implementation of the mechanical, electrical, and software subsystems. Due to time constraints, full integration was not achieved. This section details the integration of various components as well as future steps to complete the process of integration.

7.1 Electronics Rack

The electronics rack is an integration of the mechanical and electrical subsystems. Designed to allow for ease of access to multiple components, the rack has room for all of WAVE's standard electronics. As noted in the mechanical design and analysis, the dimensions of the rack take the electronics and pressure vessel into consideration. Additionally, if the electronics were to reach a certain temperature, WAVE would not be able to function reliably. Overheating would lead to catastrophic results, so the placement of components was a well thought-out decision for the integration. The placement of the electronics was also an important factor. Certain components generate more heat than others. Specific placements on the rack take this fact into account through positioning components in close proximity to the walls of the electronics housing. Other components, such as the fit-PC and the LiPo batteries, are fairly large in comparison to the other electronics. The arrangement of each component on the rack is important for the integration, as can be seen in Figure 74. The electronics rack during system testing can be viewed in Figure 75.

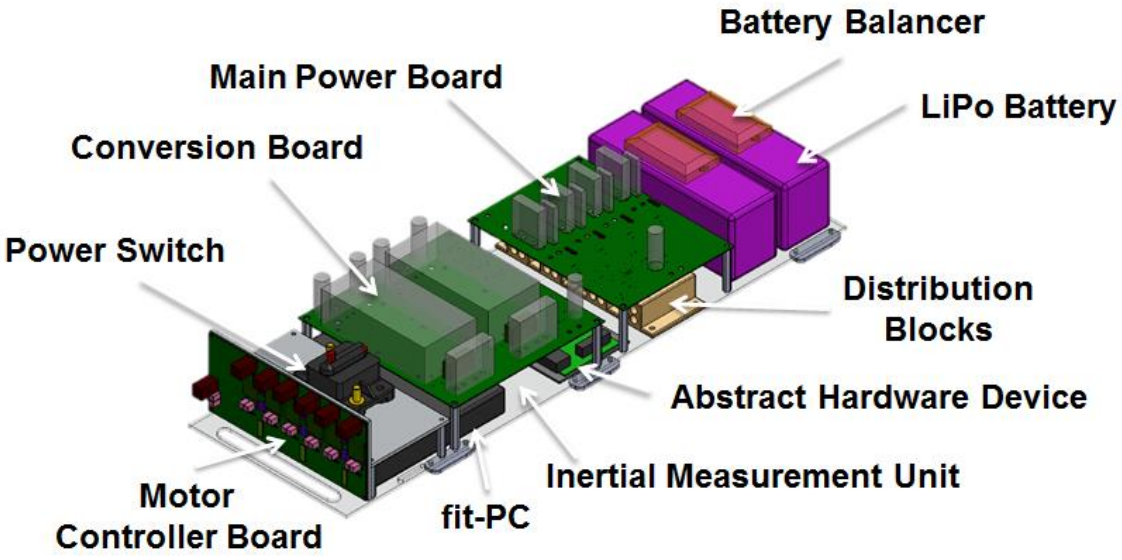


Figure 74: System Electronics Rack

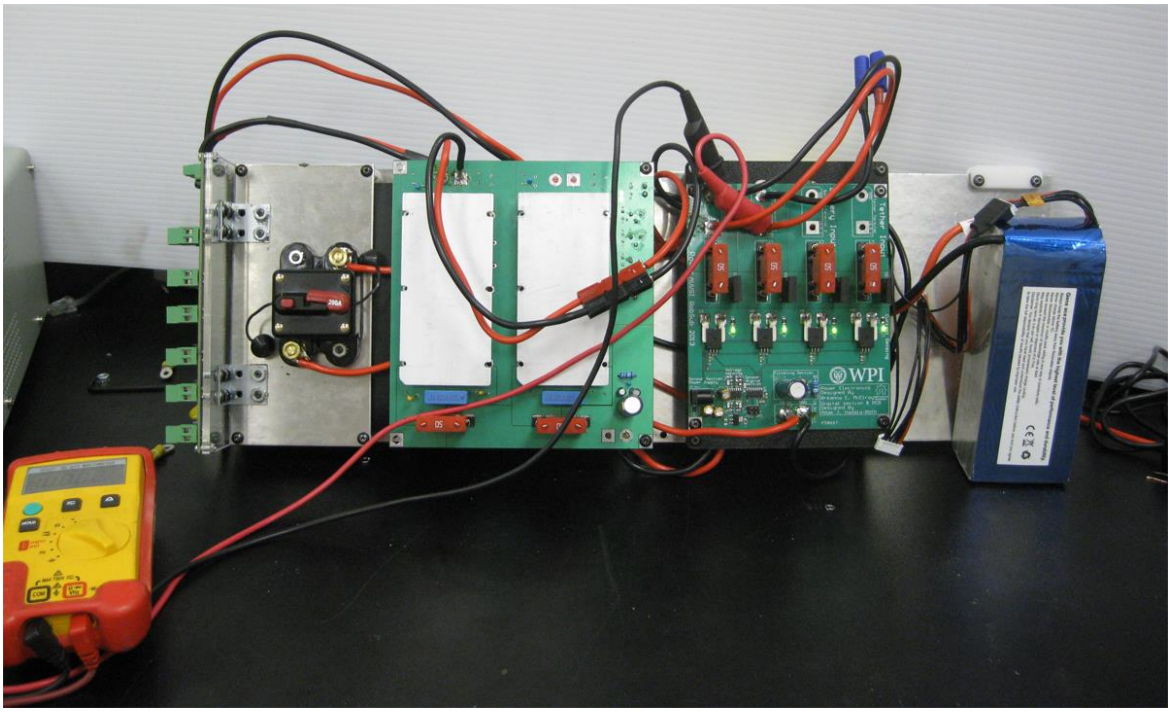


Figure 75: Electronics rack during system testing

7.2 Electronics Housing

As with the electronics rack, the electronics housing is an assembly of mechanical and electrical subsystems. Specifically, the integration comprises of the electronics rack with all boards attached,

water-proof connectors, and the rectangular housing. The water-proof connectors are located on the end-caps and are for electrical connections between the internal electronics and external modules. This iteration of WAVE planned to have external connections with the motors, ballast system, and communications tether; therefore, the ability to both communicate and distribute power between the interior electronics and exterior modules or shore-based objects is necessary, so that WAVE can be extensible by future teams.

The most important factor of this integration is the waterproofing of the housing. WAVE is designed for underwater use, and the electronics housing must protect the electronics from the platform's environment. Waterproofing is essential for the protection of WAVE's internal components because moisture could lead to dangerous situations. Moisture would be detrimental to all of WAVE's components, but the batteries are the biggest concern. WAVE requires LiPo batteries to achieve the desired level performance levels and they can become volatile if they get wet. The electronics housing allows for safe power distribution throughout the system. Due to the end-caps being removable and waterproof, the electronics racks can be removed easily without any concern about a trade-off between accessibility and safety. For additional details about the integration of this system see 10.4.3.

7.3 Communications

Communications is a combination of the electrical infrastructure with the software architecture. As outlined in previous sections, the bridge between these two subsystems is the Bowler Communications Protocol. The embedded software defines the methods for functionality in C. These methods are then sent to the high-level architecture in the form of RPCs. The high-level computing architecture then uses the RPCs to enable the desired functionality. Communication is established over USB between the AHDs and the fit-PC. Ethernet communication capability exists but was not implemented in this initial version of WAVE. The integration between embedded and high-level software was critical. Without a link

between them, the functions to drive WAVE's capabilities could not be interpreted. This communication also includes the link between sensor data and the fit-PC. Being able to receive this information allows the high-level software to interpret WAVE's surroundings and status and to plan and execute missions accordingly. These accomplishments are detailed in section 10.4.2.

7.4 Additional Integration

Time constraints restricted the full integration of WAVE, but additional steps were planned. These steps are outlined below but were not implemented.

7.4.1 Thrusters

The most important of these planned steps was the inclusion of the bilge pumps and thruster the Seabotix thrusters into the system. This step involves all three subsystems. Mechanically, each thruster would be mounted to the outside of the chassis. Electrically, the thruster leads would be connected to the motor board through the waterproof connectors. In regards to the software, the fit-PC would communicate with the AHDs which in turn connects to the motor board to drive the motors. The integration of the thrusters with the mechanical, electrical, and software subsystems allows WAVE to traverse underwater environments. This integration is an important step for developing WAVE as a research platform and competing in AUVSI.

7.4.2 Active Ballast

Like the thruster, the ballast system involved all three systems of WAVE. The tanks and their pump assemblies would be attached to the frame mirrored to each other to keep the system's center of gravity and center of buoyancy balanced as close to the true center of WAVE as possible. Further, the front ballast tank would be attached with a latch system that would allow it to easily swing out of the way of the electronics housing door. Its interaction with the power system would be a connection from the electronics housing to each motor and to the pressure sensors within each tank. Finally, the fit-PC

would use the electronics board to control the motors by receiving data from the pressure sensors within each tank, allowing it to trim the tanks as needed while underway.

7.4.3 Communications Tethering

In order for WAVE to communicate with the outside world while operating in a pool or other environment, it needs a way to send data out of the water. Due to the fact that most wireless data transmission methods are not very effective at transmitting through water, a tether is the most practical way to have this communication. A communications tether does not have to be complicated, all it needs is an Ethernet connection between WAVE and some device outside of the water. That device could be a router that then sends data to various PCs or it could be the end-user's computer directly. The two main design considerations would be that the tether itself is water-proof, as well as making sure that the tether does not get in the way of WAVE's operations. A reasonable length for the tether would be at least 9.3m (30.48ft), which is a lot of cable that could potentially get tangled or weigh WAVE down. The easiest solution would to have some sort of buoyant material that prevents the tether from sinking to the bottom of the pool and could also keep the slack of the cable away from WAVE. This tether would allow for future users to fully take advantage of the debugging capabilities of WAVE's GUI by being able to get real-time information about WAVE's status while actively operating in a pool.

8 Testing and Validation

Before WAVE's myriad of systems could be fully integrated and tested as a single entity, they needed to be tested individually to ensure that they were all functional and would not have a failure that could potentially compromise the entire robot. The following sections detail the testing of the various subsystems of WAVE and their results.

8.1 Mechanical Testing

The mechanical subsystems had to be tested and validated before they could be integrated into the whole system to ensure that they function as expected. Specifically, the subsystems are the electronics-housing pressure vessel, the active-ballast system, the structural chassis, and the overall locomotion of the craft. The testing of these subsystems is described below.

8.1.1 Electronics Housing

The electronics housing is the part of the submersible that acts a barrier between the water-sensitive electronics and the aqueous environment surrounding the sub. It is therefore an important subsystem which requires testing and validation because failure of this subsystem jeopardizes all of the electronics in the sub. The two most important failure modes of the electronics housing pressure vessel involve leaking and overheating. Other potential failure modes include, but not limited to, shorting, overloading electronic components, software crashes, and collisions with environment. The tests for the waterproofing and thermal dissipation of the housing are described below.

8.1.1.1 Waterproofing

It is paramount that the electronics stay dry, so testing of the fully-assembled pressure vessel is necessary. Assembled relevant components include the tubing, the end-caps at each end of the tubing, and the electrical connectors through the end-caps.

The testing of the pressure vessel was accomplished progressively: first empty with no connector holes, then empty with the waterproof connectors in place, and finally containing its intended hardware. One submersion test was not deemed enough to validate the waterproofing of the system. With the electronics inside the housing, the actual weight and buoyancy was taken into account. The success of these tests validated the waterproofing.

The first stage of testing was accomplished over a period of several days. The electronics housing consisting of the tube and the solid/undrilled end-caps was sealed using ratcheted straps to keep the caps in place. It was then immersed in the WPI pool at a depth of 4.3 meters for 20 minutes, agitating it every few minutes. Initially, the housing leaked during this test. Upon evaluation, the cause of the seal failure was determined to be the rough finish on the hand-machined ends of the tube. After having the ends of the tube smoothed using a CNC machine, the housing did not leak during a subsequent test. No bubbles were observed exiting the vessel and the interior was dry.

The second stage of testing occurred after the holes were machined for the connectors, and the waterproof connectors were added. The testing process was the same as the first stage of testing. The housing did not leak during the course of this test. This was indicated by putting a grey t-shirt inside the housing, near each end-cap, when the vessel was removed from the water, the shirts were still clearly dry.

8.1.1.2 Thermal Considerations

Initial, brief thermal tests were conducted after the thermal simulations produced alarming steady-state temperatures. These tests are described previously, in Section 4.3.3.2.

Due to time considerations, integrated thermal testing was not able to be accomplished. Given more time, the pressure vessel's heat transfer capabilities would be experimentally tested using the following procedure to ensure that the vessel was able to dissipate more heat than the electronics were

able to generate. The electronics are put into the pressure vessel along with temperature sensors. The vessel is then sealed and submerged in water. The electronics are powered and run for 15 minutes, the length of a standard mission. The testing is intended to simulate a typical mission, including placing load on the motors and actively reading sensor data. During this test, the temperature is monitored and recorded. If the temperature rises above the maximum temperature threshold of the contained electronics, the power to the electronics is cut, and the housing is removed from the water and opened to allow for quick ventilation.

8.1.2 Ballast

The ballast tank was subjected to several tests during the building process. Once the tank was assembled and the access hole was drilled, it was filled with water to check for leaks. All seams were then sealed with Epoxy resin and the tank was filled again, and the difference in weight between a full tank and an empty tank was measured. Next, the injection nozzle was fitted with Epoxy resin and tested for leaks at the seam by filling the tank and stopping the nozzle with a piece of crimped tubing. The tank was inverted so the water would apply the most pressure to the area around the nozzle, and no leaks were observed. A nozzle was attached to the pump and the pump was connected to a 12V motor. The 12V motor was run at a 63% duty cycle at 18.5 volts for several minutes to check both for overheating and for power requirements in running the motor. Since the motor controller is connected to the 18.5V rail, all motors have to be run at a 63% duty cycle to prevent damage. The pump assembly was then attached to the tank via tubing and placed in a large tub of water. The motor was run at 18.5V and the tank was filled until the pump could not force any additional water into the tank, which was 617mL full out of a total volume of 2059mL. During this test, the motor never drew more than 2.5A.

8.1.3 Chassis

Due to time constraints, stress testing of the chassis could not be accomplished. The greatest stress expected of the chassis is to support its own weight plus that of all of WAVE's onboard equipment, totaling up to approximately 21kg worth of mass. This happens while the craft is out of the water, particularly during the hoisting of the craft in and out of the pool. The water itself will distribute stresses on the frame evenly so that the stress while submerged is assumed to have a negligible effect.

Given more time, testing would have followed this procedure. To ensure that the chassis can in fact support the expected forces the chassis is loaded up to the appropriate amount of afore-mentioned weight. Also the chassis is suspended, using cables from all four corners to a single attachment point. This test will simulate putting the vessel in the water using a crane.

8.1.4 Locomotion

Due to time constraints, this testing was not able to be completed. Given more time, the testing would follow the following procedure. To ensure that WAVE can travel at the design specification's minimum 0.5 m/s velocity the chassis (complete with electronics housing), thrusters, and ballast systems would be put into the water, in accordance with Appendix D: Waterside Deployment and Recovery SOP to have its locomotion measured with a stopwatch over a known distance. In this test WAVE is driven at full speed past one swimmer on one side of the pool and kept on a straight course until it passed another swimmer at a known distance away from the first swimmer. This is made easier by using the patterned tiles lining the WPI pool. When the swimmers see the front of the UUV pass by them on their perpendicular, they immediately signal the person on shore with the stopwatch. This is also made easier by lining up perpendicular to the patterned pool tiles. The first and second swimmer's signals indicate that the chronometer be started and stopped, respectively, recording the time. The swimmers leave distance between themselves and the sub in order for the sub to accelerate and decelerate in order to

gain a meaningful measure of cruising speed. The above exercise is performed 10 times and the clocked times averaged in order to have a good estimate of the time it takes WAVE to cover the known distance. By dividing the averaged time by the known distance the cruising speed of WAVE can be determined, which will either meet or not meet the 0.5 m/s forward velocity requirement.

The above exercise can be further performed on the lateral and vertical axis in order to have known travel speeds along those axes. To measure travel time along the vertical axis, start WAVE at a known depth, like the bottom of the pool, and time its ascent to the surface.

8.2 Electrical Testing

The electrical subsystems underwent testing before the planned integration with WAVE. The platform required the verification of the AHDs, power system, sensor suite, and embedded software before full system assimilation could be possible.

8.2.1 Testing the AHD

Testing of the Abstract Hardware Device was a long and tedious process involving several stages of surface mount soldering with different tools leading up to the last step of testing programming over JTAG. Testing the AHD is broken down into multiple sections: populate onboard power, populate microprocessor and associated supporting circuitry, JTAG test, populate and verify USB, populate and verify Ethernet, and lastly add all female pin headers. The reason for breaking the AHD testing into discrete tests goes along with the concept of getting a board up in running in steps. Printed circuit boards like the AHD are essentially massive circuits with many components wired in parallel. Each of the sub circuits within this massive parallel circuit either are dependent on one another or can affect one another either negatively or positively. In the interest of making hardware debugging easier, the assembly of the board is broken into steps, starting with the power supply and going up from there. In the case of the AHD, the types of components used were nearly all surface mount components of

varying package type. To solder these components to the board a hot air reflow station was required. A hot air reflow station consists of a hot air gun, foot controlled solder paste dispenser, and usually a stereo microscope. Using the tools part of the reflow station the AHDs were fully assembled.

Assembling and testing the power supply is the first step because every other section of the board needs power to function and in the case of the AHD cleanly filtered and highly stable power is required to ensure stable operation of the ARM microprocessor aboard the AHD. The power section of the AHD is almost all surface mount with the exception of two input filter caps and the DC barrel jack. When assembling the power section the first step is to solder on the input Schottky diode, and the two regulators for 3.3v and 5v along with their bypass and damping capacitors. After the regulators are attached the section must be tested by connecting the power input to a bench-top power supply and monitoring the output of each regulator with a volt meter. If the voltages are around where they are supposed to be then assembly can continue. If not then the power section must be checked for either shorts or improperly soldered components. In the case of AHD 1.0 the dot indicator on the Schottky diode did not the PCB so it had to be flipped otherwise there was a short circuit. When assembling the power section of AHD 2.0 there were no such problems as the Schottky diode used on 2.0 was a different type.

Moving on with the AHD assembly the next step was to attach the microprocessor and components that it needs to function. The microprocessor being used is the NXP LPC4337JBD144 which is a thin quad flat package (TQFP) with 144 pins. This sort of package requires different soldering tools not found in the reflow station setup. For soldering the 144 pin TQFP a UV lamp reflow station was needed. This sort of tool uses a UV lamp to melt the solder attaching the component to the PCB. A UV lamp was required because there are 144 pins to attach simultaneously and it's important for them to all be solidly attached. A UV lamp reflow station was available for use in the Sensitive Robotics Lab in

Fuller Laboratories. Using the UV lamp the microprocessor was soldered onto the AHD. Next the supporting circuitry of the microprocessor was populated, this includes crystal oscillators and their load capacitors, pull-up resistors, reset circuitry, and bypass capacitors.

After the microprocessor and supporting circuitry is added to the PCB with the already functional power supply section the next step is to populate the JTAG interface section then test if the board can connect to the JTAG programmer. The JTAG programming interface consists of a 1.27mm pitch 10 pin header (2X5) and pull-up resistors on the two inner most pins. In the case of AHD 1.0 the JTAG connector was a surface mount model so care had to be taken to solder it without melting the plastic parts but for AHD 2.0 the JTAG connector was through hole and only the resistors were surface mount.

At this stage everything required to test the core functionality the AHD has been soldered on to the PCB. Before moving further with assembly it is important to test LPC4337JDB144's ability to interface with the JTAG programmer. If it cannot interface at this stage there is either a problem with soldering, an improper JTAG circuit or damaged components. When AHD 1.0 met this stage it was unable to program over the JTAG interface. When AHD 1.0 met this stage it was unable to program over the JTAG interface. Upon closer examination of AHD 1.0, looking through the data sheet, and reviewing the reference design (LPC4330-Xplorer Schematic) it was found that AHD 1.0 was lacking bypass capacitors on all voltage input pins and the JTAG interface was lacking pull-up resistors. In an attempt to get AHD 1.0 to program the PCB was modified with external pull-up resistors on the JTAG interface, and the power section was removed and replace with wires that connected the 3.3v rail to a bench-top power supply.

After many attempts of trying to get AHD 1.0 to program it just wouldn't work. All signals were examined with an oscilloscope, all JTAG signals were making it to the correct pins but still the LPC4337

would not connect to the JTAG. Next the clock signals were checked but with ARM Cortex M4's the chip does not start up the crystal oscillators and internal clock circuitry until a JTAG link is established. One of the first things that happen at the start of a JTAG link is that the processor is configured for how it is to run. The configuration for the processor is set in a C header file called systconfig.h; in this file the max clock speed is set and the sources of all clock signals for the core clock and peripherals are defined. Since the JTAG link was not ever successful with AHD 1.0 there weren't any clock signals when examining the crystal circuitry. In the end AHD 1.0 was unable to program over JTAG. In the field of embedded computing and especially in regards to PCB design this usually indicates that the board will not work correctly even with external modifications.

Since AHD 1.0 was unable to pass the basic test of JTAG program validation a new design was required. AHD 2.0 was almost a complete redesign compared to AHD 1.0. The changes in AHD 2.0 included a JTAG section with pull-up resistors on each of the four lines, retraced USB section with equal length differential pairs for the two data lines, reroute of the entire Ethernet section following the equal length differential pair design guidelines in a stricter manner, Arduino Leonard breakout header spacing instead of Duemilanove, additional digital IO breakouts, removed PWM coprocessor (using pins that support the state configurable timer for PWM instead), and bypass capacitors of 100nF were added to all voltage inputs on all devices. When the redesign of AHD 2.0 was completed it was sent out and manufactured by Advanced Circuits. After 3 days of waiting the new boards were ready to undergo testing and validation.

Assembly of AHD 2.0 followed the same rules as AHD 1.0, starting with the power section, then moving on to the microprocessor and, lastly, the JTAG section before the crucial JTAG test. After AHD 2.0 was populated to the same degree AHD 1.0 was before the redesign it was time to test JTAG programming. The first few attempts of JTAG programming AHD 2.0 were unsuccessful. Since AHD 2.0

was following the design specifications laid out by ARM and NXP it was puzzling as to why it was not working. Before moving forward the board was tested for short circuits by examining every single component underneath a stereo microscope. The LPC4337 was replaced thinking that the first one may be dead from a short, and the ceramic capacitors on the power section were replaced with tantalum equivalents to ensure more stable operation of the voltage regulators. After all the modifications the AHD 2.0 still refused to program over JTAG. The AHD JTAG testing validation setup can be viewed in Figure 76.

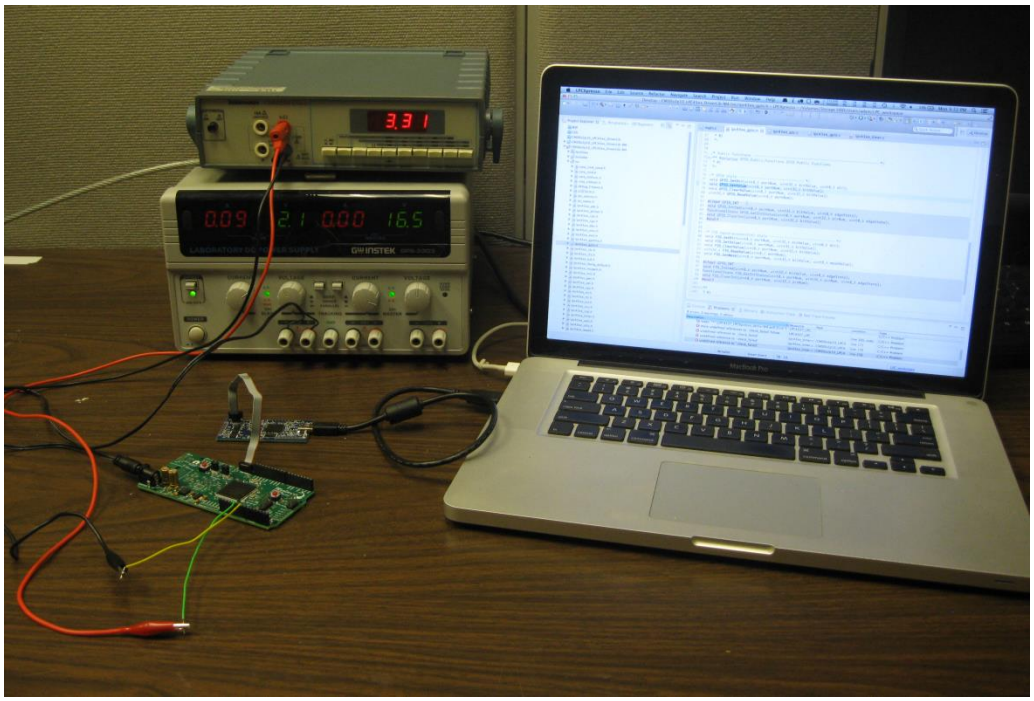


Figure 76: JTAG Testing Setup

In a more extreme and aggressive tactic one of the two spare PCBs of the AHD 2.0 design was populated from scratch being examined under the microscope closely every step of the way. Even the fresh AHD 2.0 board still refused to program. Looking in the datasheet and comparing it with what NXP tech support said about JTAG pull-up resistors, it was found that only the TCK and TDO JTAG wires needed pull-ups and that pull-ups on the other lines may cause problems. The extra pull-ups were

removed. Still the AHD 2.0 would not program. Upon closer examination using a volt meter to manually check every connection it was found that the reset button was wired incorrectly. Given the way in which it was laid out on the board it was acting as a short to ground holding the LPC4337 in reset permanently. The next step was to remove the reset button and attempt to program. After the reset button was removed the AHD 2.0 was connected to the JTAG and a programming routine was instantiated in LPCXpresso which ended in success.

Since the new AHD design was able to program it was safe to move forward with assembly and testing. When the microprocessor is able to be programmed it's best to test the functionality of lower level peripherals before testing higher level peripherals such as Ethernet and USB in this case. After achieving success in JTAG programming, the GPIO driver function library was tested for reading and writing to digital IO pins with great success. At this stage the project was coming to a close so not every feature was tested on the AHD. Despite the lack of time to complete thorough testing procedures, the AHD's ability to be able to be programmed and debugged over JTAG and run code is a sign that the design is likely to be stable and functional.

8.2.2 Power System Testing and Evaluation

The Power System testing involved four primary areas of validation. Familiarity needed to be gained and confirmed with the battery charge and discharge curves. The main power supply board needed the voltage and current sensing to be evaluated. Lastly, the conversion board needed to have the 12V rail confirmed to be as expected.

8.2.2.1 Battery Testing

The power system is designed to distribute the power from the battery inputs throughout WAVE. The batteries provide a nominal voltage of 18.5V, which is the rated voltage. When the battery is fully charged it has an open circuit voltage of 21V and when fully discharged it has a voltage of 15V. Figure 77

and Figure 78 show the discharge curves of the battery when sourcing approximately 10A for 70 minutes. The voltages of the individual cells were recorded from the battery management unit display and the battery voltage output value was measured by a digital volt meter. From the two charts, it can be seen that the cells in series do add up to the measured battery voltage as expected with a variance of ±60mV between the measured output and the individual cell sum, which is acceptable. A high power resister of 2Ω was used to discharge the battery and can be seen in Figure 79.

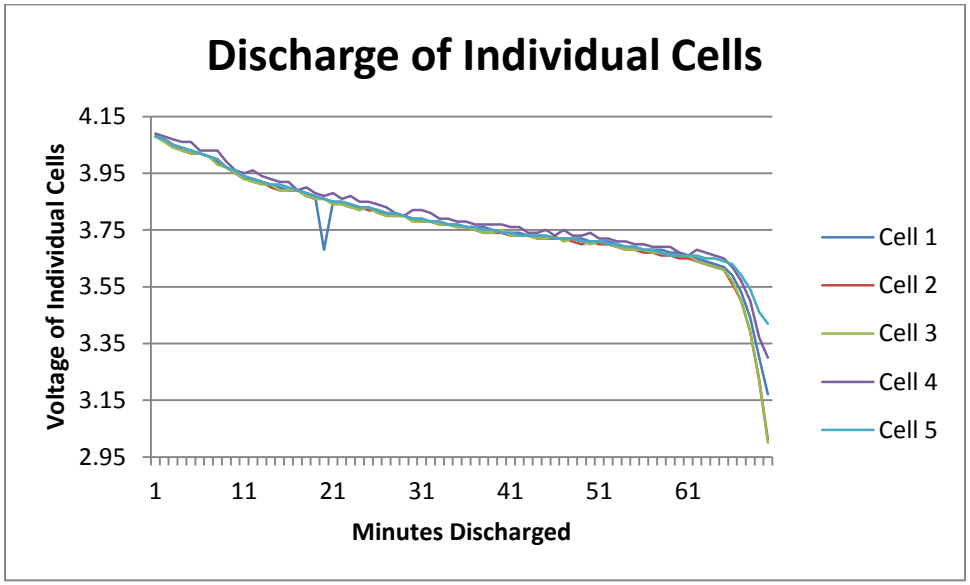


Figure 77: Discharge of Individual Battery Cells

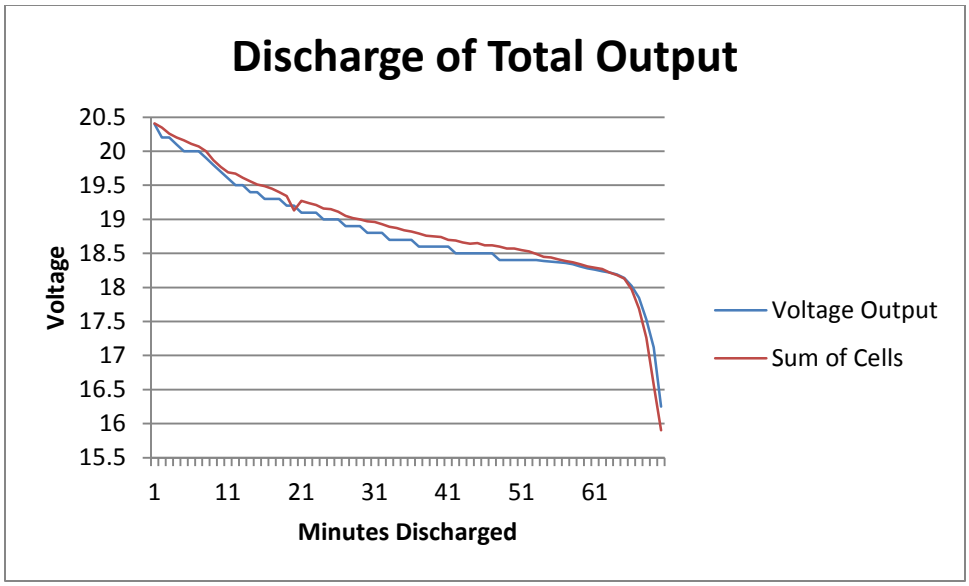


Figure 78: Discharge of Battery

Figure 79: High Power 2Ohm Resistor

After the discharge test, the battery was charged again and the curve was mapped using LogView, which is compatible with the charger. The charger has a safety feature built in that ends the charging cycle after 5000mAh have been put into the battery. As the capacity of the battery being used is 10,000mAh, the charging ended in the middle of the charging curve and was then started again. Figure 80 shows the voltage of the individual cells vs. time and Figure 81 shows the total voltage vs. time for the first half of the charge cycle. These charging curves behave as expected. As can be seen in Figure 80 the batteries were discharged to approximately 16.5V, which is where the charging curve begins. Provided 5A for one hour, the battery was charged to 19.5V as expected.

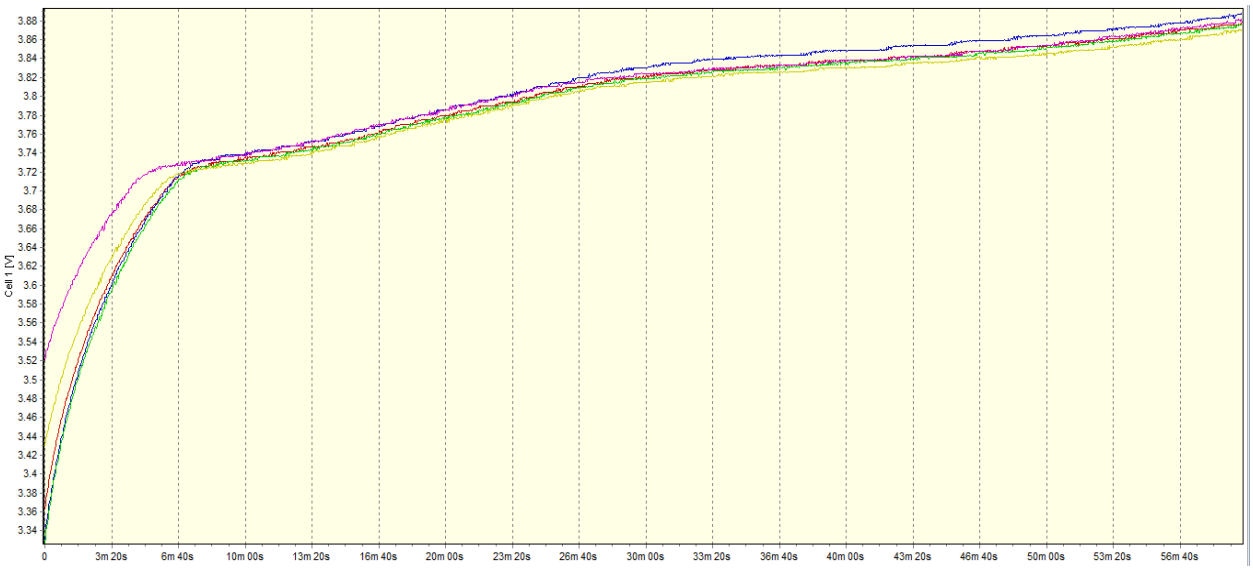


Figure 80: First Half Charging of Individual Cells

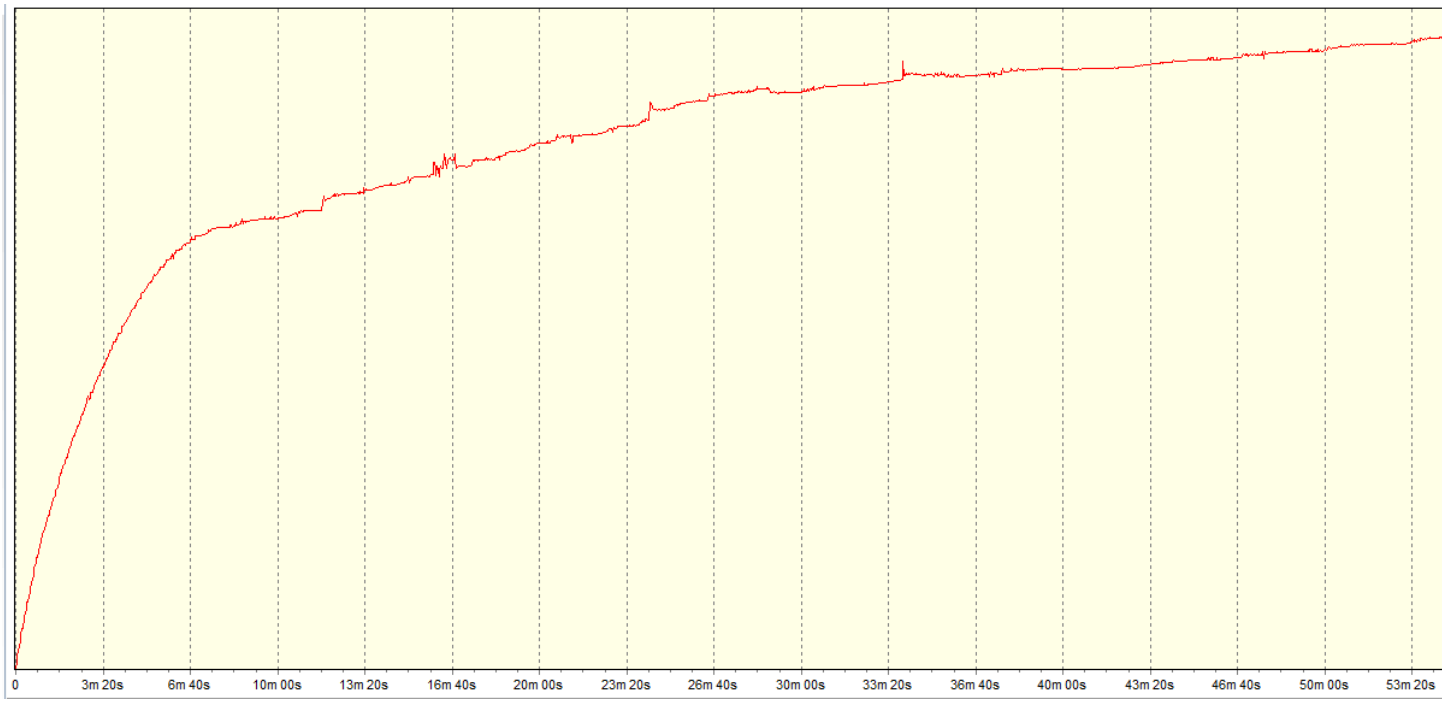


Figure 81: First Half Total Voltage Output

Figure 82 shows the voltage of the individual cells vs. time and Figure 81 shows the total voltage vs. time for the second half of the charge cycle. As can be seen in Figure 81 the battery was charged to approximately 19.5V, which is where the charging curve begins. Provided 5A for the remaining charge cycle, the battery was charged to the full capacity of 21V over 56 minutes as seen in Figure 82 and Figure 83.

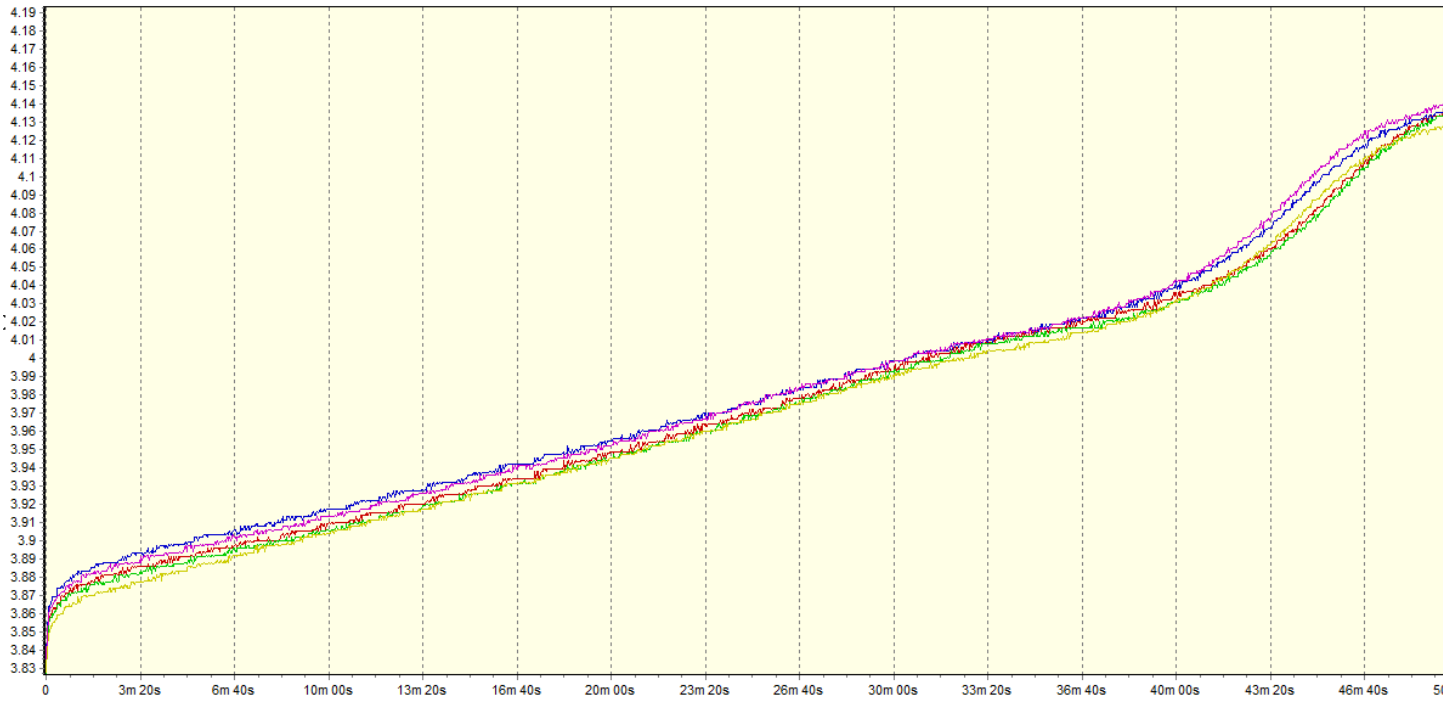


Figure 82: Second Half Charging of Individual Cells

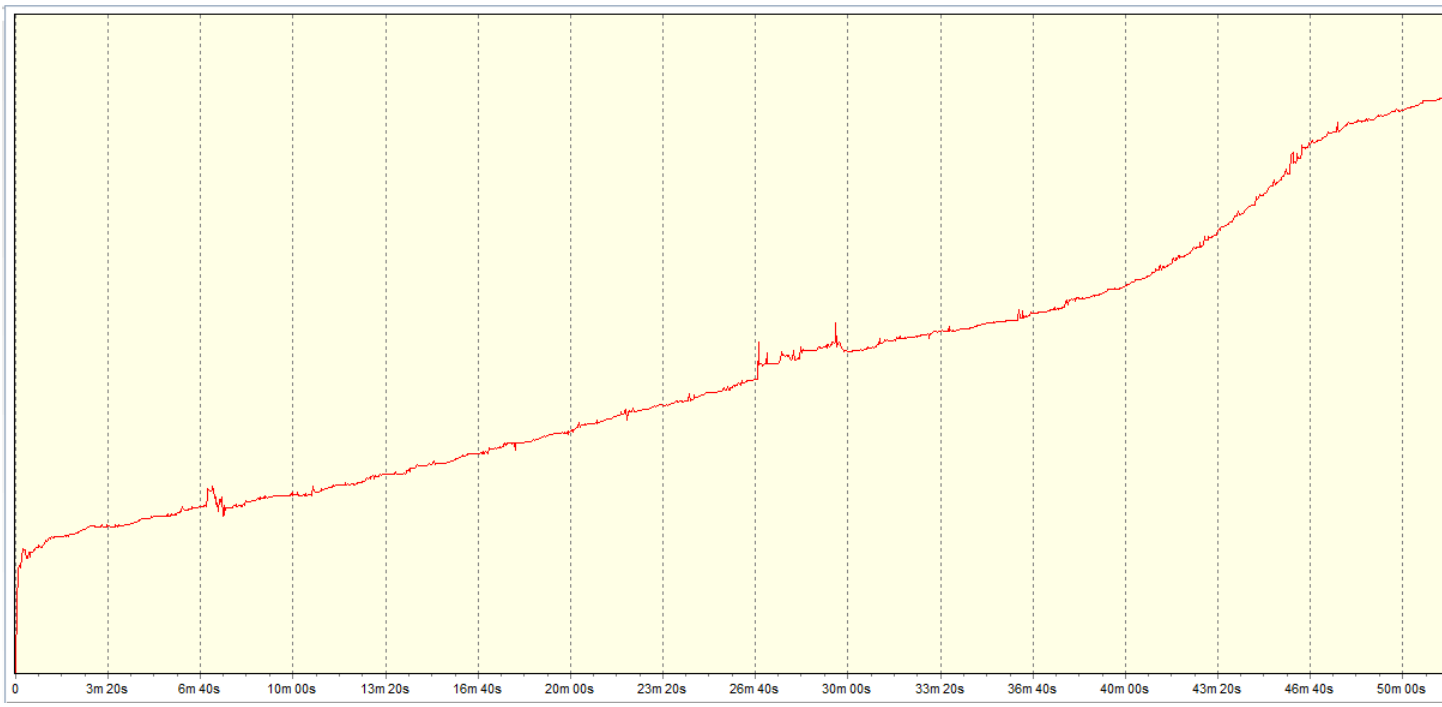


Figure 83: Second Half Total Voltage Output

8.2.2.2 Current Sensor Testing

Allegro ACS259 Hall IC current sensors were supplied by Allegro Sensors. They were first tested by using a supplied ACS259 development board connected to a power supply powering a network of 8 Ω, parallel resistors with switches that either add the resistor to the parallel network or open it from the circuit to exclude them from the network as can be seen in Figure 84.

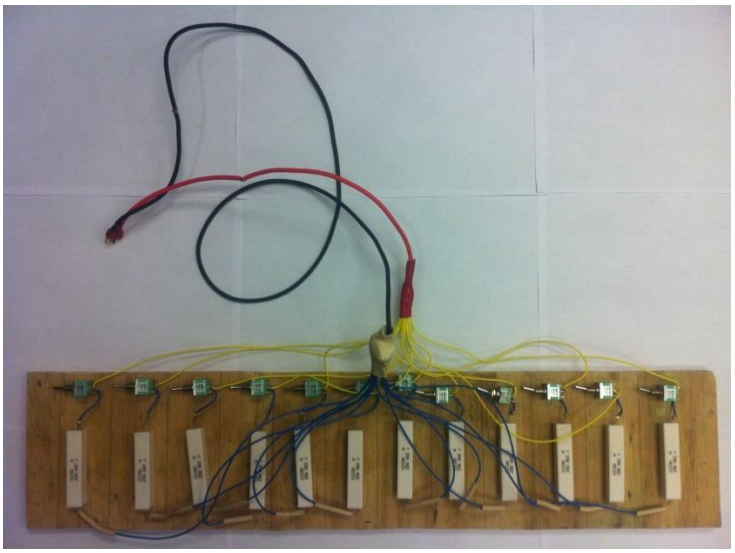


Figure 84: Network of Parallel Resistors

The higher current sensing Allegro Sensor ICs are generally bidirectional. Though this functionality is not utilized for WAVE, the sensor's high current sensing capability with low power loss was optimum for WAVE's power system. The sensor is rated to sense ±150A and has a maximum common supplied voltage of 3.6V, but has an absolute maximum supplied voltage of 8V. The testing was performed by powering the sensor from a DyIO which provided a 5V power supply, causing the ±150A to be mapped from 0V to 5V. When no current passes through the sensor, an output of 2.5V is expected. This means that 2.5V – 0V maps the current value in the negative direction and 2.5V – 5V maps the current value in the positive direction. A total of 2.5V is available to represent 150A in one direction, yielding a conversion factor of roughly 16mV/A. In addition to powering the sensor, the DyIO was used to read the voltage output signals.

Table 15 shows the sensing behavior given a 5V input into the network of parallel resistors and Table 16 shows the same set up, but with a 19V input. The signal out differs the greatest from the expected value with a difference of 70mV and is underlined in Table 16. This 70mV divided by the 16mV conversion factor, concludes that the current sensor has an accuracy of ±4.375A. Note that for both

tables the resistance is the actual measured resistance. The current sourced is the actual recorded value from the display of the power supply, which was used for reference as these values were not in accordance with Ohm's law. The expected signal out is calculated by multiplying the current sourced by the 16mV factor and adding it to 2.45V as was the measured 0A signal output. Also, the signal out is the value recorded from the DyIO NR Console GUI.

Table 15: Current Sensing 5V Input

Resistance (Ω)	Current Sourced (A)	Expected Signal Out(V)	Signal Out (V)
Open	0	2.45	2.45
8	0.7	2.46	2.45
4	1.3	2.47	2.45
2.66	1.8	2.48	2.45
2	2.4	2.49	2.45
1.6	2.9	2.5	2.45
1.33	3.4	2.5	2.45
1.14	3.8	2.51	2.5
1	4.3	2.52	2.5
0.89	4.7	2.53	2.5
0.8	5.1	2.53	2.5
0.73	5.6	2.54	2.5
0.67	6	2.55	2.5

Table 16: Current Sensing 19V Input

Resistance (Ω)	Current Sourced (A)	Expected Signal Out(V)	Signal Out (V)
Open	0	2.45	2.45
8	2.4	2.48	2.45
4	4.6	2.52	2.51
2.66	6.7	2.56	2.51
2	8.7	2.60	2.56
1.6	10.6	2.62	2.57
1.33	12.5	2.65	2.62
1.14	14.3	2.68	2.62
1	16	2.71	2.64
0.89	17.6	2.73	2.67

8.2.2.3 Voltage Sensor Testing

The voltage sensing section uses an op amp in conjunction with a voltage divider using a 174k Ω and 1M Ω resistor, providing a conversion factor of 0.148 multiplied by the battery voltage. The circuit shown in Figure 85 was tested on a bread board and worked as expected, however; the surface mount resistors that were ordered did not match the footprint on the board. To compensate, 165k Ω and 1M Ω resistors were found on campus and were used on the actual PCB, giving a conversion factor of 0.142.

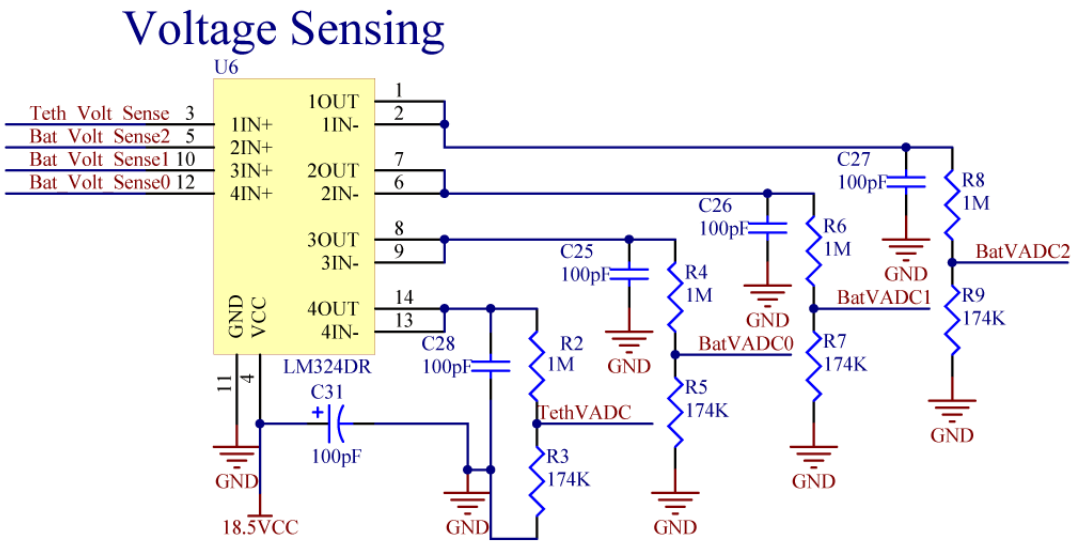


Figure 85: Voltage Divider for Voltage Sensing

The results of the voltage sensing test on the PCB can be seen in Table 17 where the 165kΩ and 1M measured values are the actual, corresponding measured resistances. The expected value was calculated using the actual measured results for the conversion factor and multiplied by the 18V input. The measured value is the actual value measured by a digital voltmeter.

The measured value did not correspond to the expected value and had an equal value for every battery input. The test was configured with only one power input, but the same voltage was displayed on each voltage sensing rail. There should only be one output on the corresponding input rail and the

other three voltage sensing dividers should output 0V. As can be seen in Table 17, this was not the case, which led to the discovery that the current sensors are shorting the voltage sensing inputs.

Table 17: Voltage Sensing Testing Data

	165kΩ Measured	1MΩ Measured	Expected Value (V)	Measured Value (V)
Battery 1	165.0k	1.18M	2.576	0.59
Battery 2	164.4K	1.23M	2.476	0.59
Battery 3	165.4K	1.18M	2.582	0.59
Power Tether	153.2k	0.99M	2.814	0.59

Figure 86 and Figure 87 show the schematic for the current sensors section and its outputs (labeled Bat_Sense_1 etc.) that then go into the voltage sensor. Figure 87 shows, in red, how the voltage rails are actually connected and how the inputs to the voltage sensing section are all actually shorted together. This causes the battery rails to be combined in parallel before the voltage sensing occurs on the board, giving each battery sensing input the same value. It is unknown why the value of 0.59V was obtained as this was far from the expected value. Additional testing would need to be performed to understand this behavior. The board would need to be redesigned to have the capability to monitor individual battery voltages.

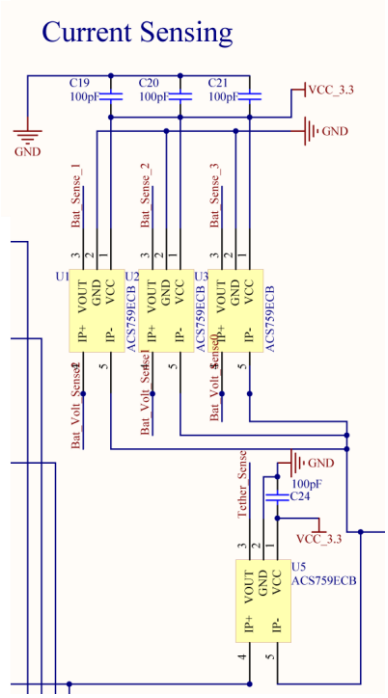


Figure 86: Current Sensing Configuration

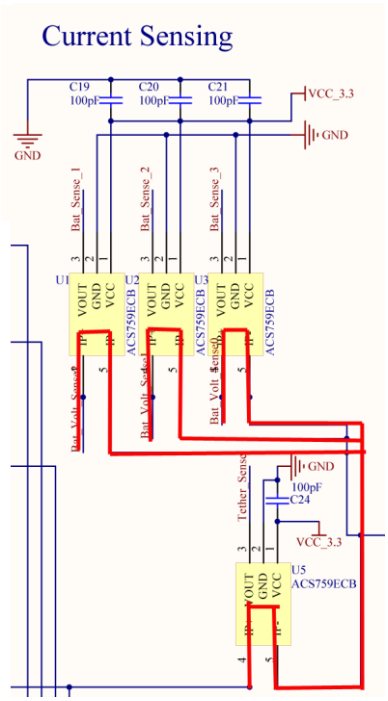


Figure 87: Current Sensing Configuration Shorts Voltage Sensing Inputs

8.2.2.4 Conversion Board Testing

To test the conversion board, a power supply was connected to the board with a 5MΩ resistor connected as the load. The conversion board did not work as expected, but would output either 0V or a voltage between 3V-9V and would make an audible buzz. During debugging, it was found that the converter was implemented incorrectly in two ways. First, the foot print of the converter was labeled backwards in the schematic. Second, the grounding of the device was misunderstood and therefore, connected incorrectly. Figure 88 was referenced when making the schematic for WAVE, which can be seen in Figure 89.

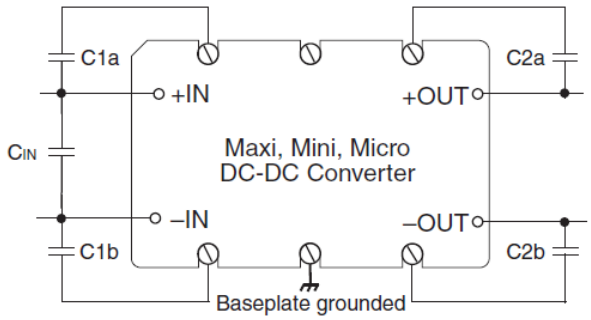


Figure 88: Typical Vicor Converter Configuration

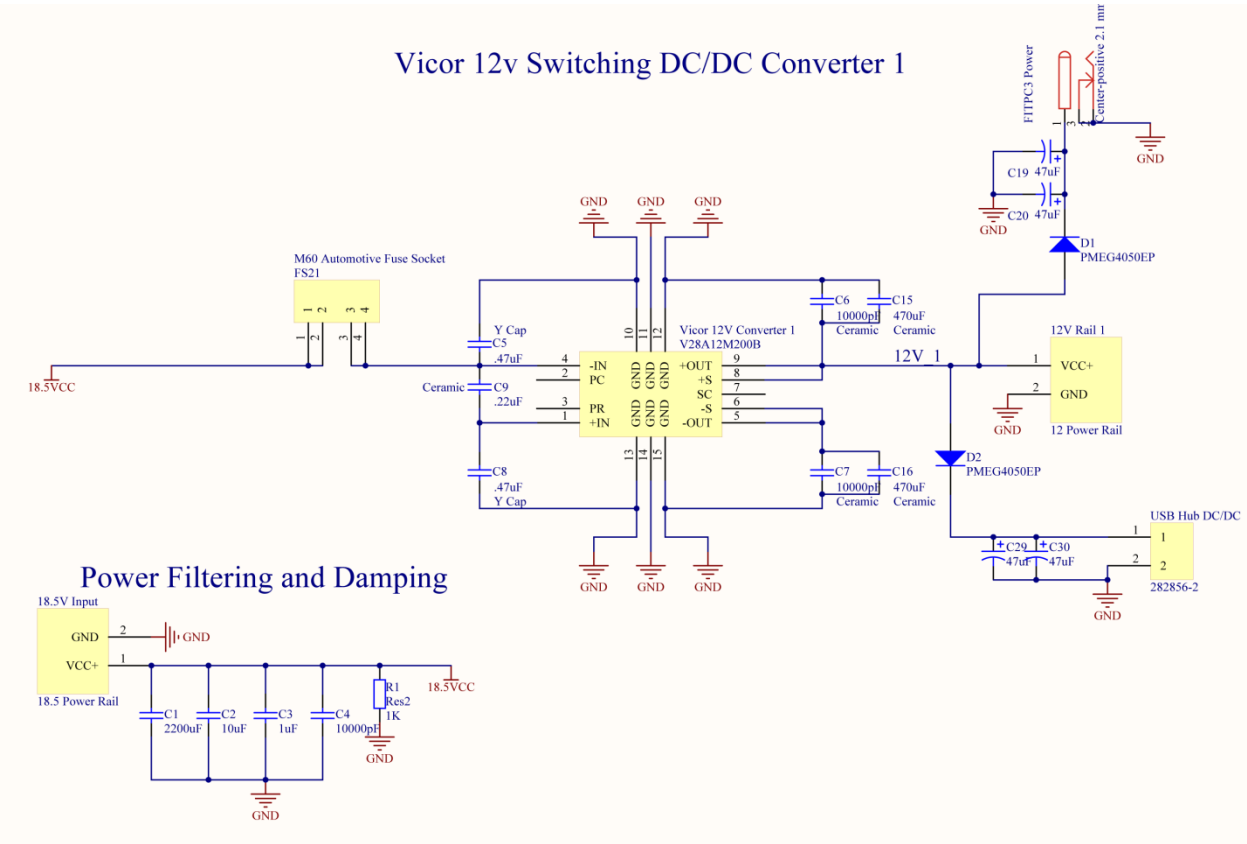


Figure 89: Conversion Board Schematic

In the diagram in Figure 89, the middle, bottom screw securing the converter to the PCB board is grounded. From this, it was incorrectly assumed that because the baseplate is conductive, each screw should be connected to each other and to the ground plane on the board. However; the earth ground symbol is used and in this instance is intended to represent that the baseplate should be isolated from the system.

Figure 90 shows how the converter is actually configured on the inside and shows a transformer being implemented. It is known that the negative input and output of the transformer should be directly grounded and isolated from one another. With these design errors, modifications to the board had to be made and can be seen in Figure 92.

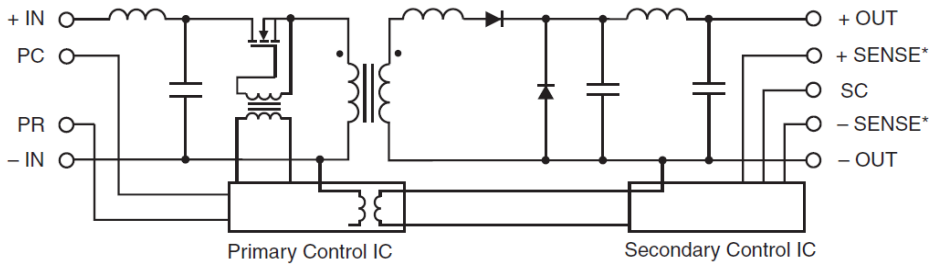


Figure 90: Inside the Vicor Converter

8.2.2.4.1 Conversion Board Modification

To modify the design, first the converters were flipped. As the footprint was laid out backwards in the schematic, it was in turn, laid out correctly when applied to the bottom side of the board. The capacitors located at C7, C15, and C16 in Figure 91 were removed. The screws were insulated so that they no longer touched their ground pad, allowing the base plate of the converter to be completely isolated from the circuit. The negative input pin has a modified wire connecting it to the ground of the 18.5V rail distribution block and the negative output pin has a modified wire connecting it to the nearest screw terminal ground on the board. Both of these modifications are shown in red in Figure 91 and can physically be seen in Figure 92. The black "X's" in Figure 91 mean that with the modification, there is no longer anything directly connected to that solder pad.

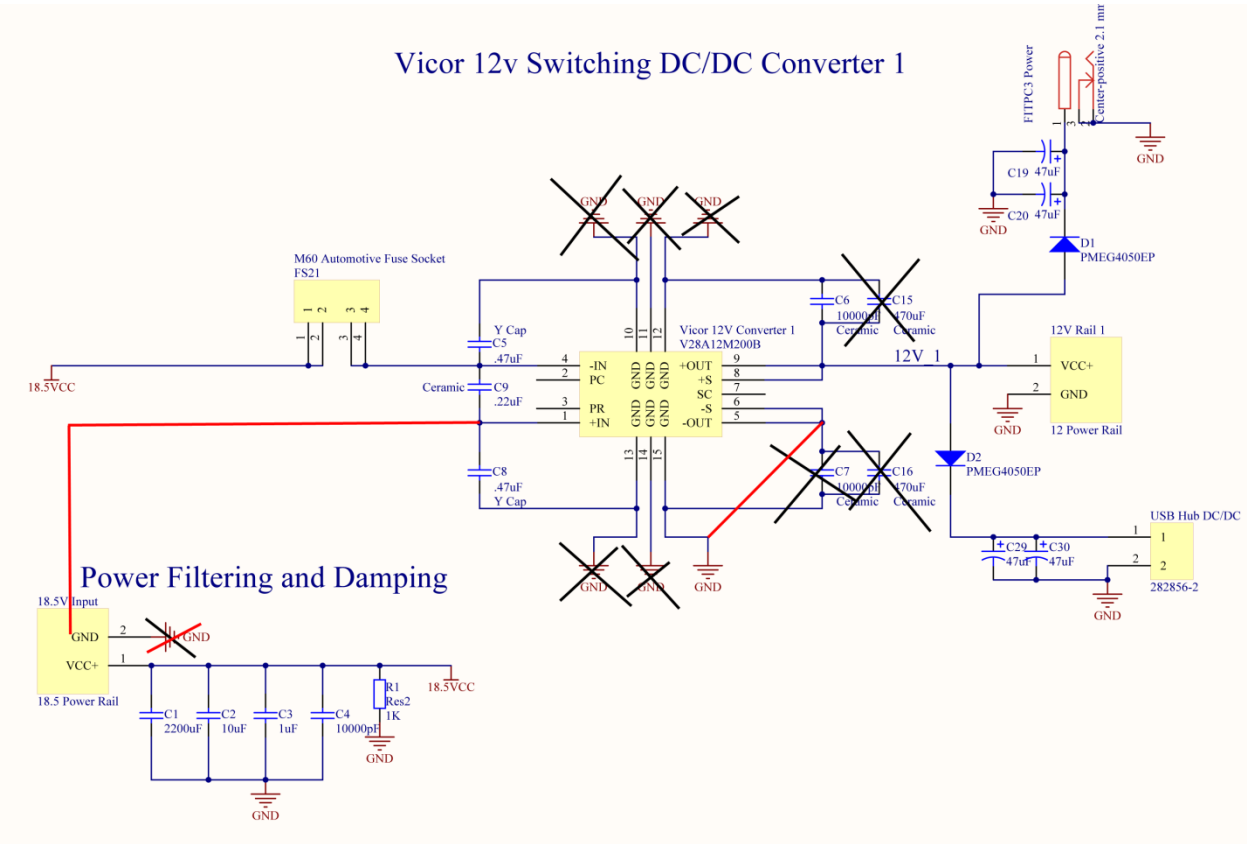


Figure 91: Modification of Conversion Board (Schematic)

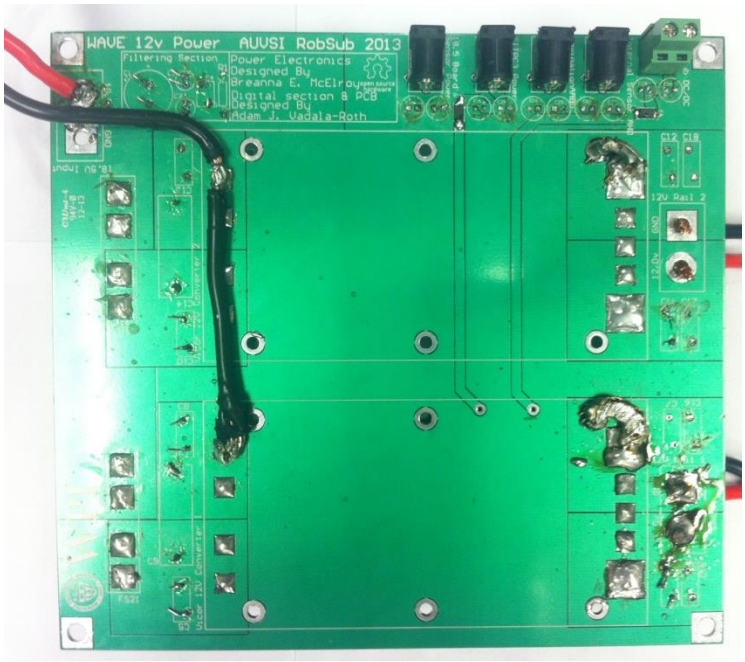


Figure 92: Modified Conversion Board (PCB)

8.2.2.4.2 Modified Conversion Board Testing

The conversion board was designed to take the main supply rail from the main power board and convert it to a clean 12V signal. This 12V rail is then used to power the Fit-PC, AHDs, and any other 12V component that may be added in the future. Of these, the fit-PC is the most sensitive to power input and can be powered between 10V-16V. The conversion board was first tested with a power supply swept from 16V up to 22V and the output maintained exactly 12V throughout the entire sweep. Then the board was connected to the main power board, powered by one battery, and given a load of 5MΩ. It provided an output of 11.99V, which is well within the fit-PC powering capabilities. This was a positive result, but only 2.4µA were being sourced. A closer look at the voltage output with various currents was considered and the results can be seen in Table 18, where the voltage input was 19V, the "Current Sourced" is according to the power supply display, and the "Measured Out" is the output voltage of the conversion board measured with a digital voltmeter. As expected, the output voltage was steady as the current changed.

Table 18: Conversion Board Output with Current Considered

Resistance (Ω)	Current Sourced (A)	Measured Out (V)
Open	0	11.99
8	1.3	11.99
4	2.3	11.99
2.66	3.3	11.99
2	4.3	11.99
1.6	5.3	11.99
1.33	6.3	11.99
1.14	7.3	11.99

8.2.3 Sensors

Each of the individual sensors needed to be tested to confirm functionality of the sensor suite. The capabilities of the flood sensor, pressure sensor, and temperature sensor each needed to be validated. The following subsections outline these tests.

8.2.3.1 Flood Sensor

The first test that was performed with the flood sensor was a test to determine how deep the flood sensor had to be submerged in order to show detection of water. In order to do so the flood sensor was set up as shown below in. The output of the flood sensor was displayed on a serial monitor after interfacing it with an Arduino Uno. When tested it was determined that when the dome is completely submerged in water the output changes from logic high to logic low. As the flood sensor must be mounted upward this would be at a depth of 22.49 mm. At this level some of the electronic components which are attached to the floor of the electronic rack would already be submerged in water before communicating this information back to the GUI. As a result it is suggested that the flood sensor design could be improved through the use of probing wires or a shorter dome that would detect water at a lower level.

8.2.3.2 Pressure Sensor

For the pressure sensor following tests were designed but not implemented: power up test, sensor full submersion Test, pressure test in air. For the first test the pressure sensor would be powered with 10 V

using a bench top power supply and the voltage output would have been measured using a digital multi-meter. This would be used to show functionality. After it was ascertained that the pressure sensor was functioning properly then the next test to perform would have been the full pressure sensor full submersion test. In this test the pressure sensor would have a marking along the cable denoting different depths. As the pressure sensor was submerged to different depths the output readings would then be recorded. In order to verify if the correct depth corresponded to the reading the following calculations would have to perform. Depth conversion to psi to ensure that it is well within the operating range of the pressure sensor, after this is verified a conversion factor can be formed between the voltage output and the depth.

8.2.3.3 Temperature Sensor

The temperature was not able to be tested on other components within the vessel due to time constraints. Although in order to ensure that the data value from the temperature sensor were correctly identified with the correct value of temperature. The temperature sensor was compared to the value of a calibrated thermocouple and found to be accurate to +/- 2 degrees C.

8.2.3.4 Humidity Sensor

The humidity sensor was tested via breadboard design with the Ardunio microcontroller. It was not possible to find calibrated humidity sensor in time to compare with current outputs of the system.

8.2.3.5 Integrated System Test

The integrated sensor test involved providing a breadboard design of the system and testing the system functioning as a whole. In order to perform this test the system was first bread boarded. After the system had all the elements connected then it functionality of each sensor then had to be tested. When the system was all powered up the first functionality to be tested was the temperature sensors. Initial findings found when connected to the Arduino Uno in order to give temperature outputs the outputs

were not displaying as connected. As such the next step was to go back and re-verify the connections. When this was done it was found that there were some wiring problems with the SDA and SCL line via the I2C bus extenders after these wiring problems were corrected and the temperature sensors were reconnected to the Arduino Uno the appropriate temperatures than began to display. The next step was to verify that the flood sensors were working within the system as well. This was also done by interfacing the sensor with the Arduino Uno in order to test whether the output was correct in the presence of water. This test proved successful. The next test was interfacing the humidity sensor with the system via the Arduino Uno. This test proved successful as well. Complication occurred when the humidity sensor and the temperature sensor were trying to read out on the same I2C address. This was due to the calling of the wrong address for the humidity sensor. Some problems encountered within this system was the fact that due to the extensive wiring sometimes there were delays in information and that it was not possible to receive a constant value. In addition the bypass capacitors within the system also caused an issue taking long periods of time to discharge and as a result affecting the operation of different sensors. As such in the future it would be advantageous to add circuitry to discharge the capacitors at a faster rate and choose lower values for bypass capacitors.

8.2.4 Low-level Computing

The embedded system needed to be verified to ensure the functionality of the software. The ability to download code and communicate was critical.

8.2.4.1 JTAG download confirmation

Before the LPC 4330 dual core processor could be incorporated into the Abstract Hardware Device, the functionality of the M4 core needed to be confirmed. The NGX LPC4330-Xplorer board was obtained to run tests on the M4 core. NXP provides several example projects for LPCXpresso, which simplified the testing process. The first example project used was entitled Blinky. While this project was very simple, the example proved to be an important test of the M4 core's capabilities. There were four goals of this

test: compile code in LPCXpresso, download code via JTAG, communicate with the M4 core, and perform a simple task. In the case of Blinky, the simple task was to alternate flashing the two LEDs on the Xplorer board. As this example project was ready to work out of the box, the only challenge posed was to correctly link the project to the M4 core driver library. After integrating the driver library into the include directory of the project, the code compiled. The project was then downloaded over JTAG and was able to communicate with the M4 core. Once the download process was complete, the LEDs on the Xplorer board did in fact alternate blinking. This test successfully confirmed the four goals. For all future embedded testing, test code would be downloaded via JTAG to the M4 core.

8.2.4.2 Echo Server

As stated in previous sections, Neuron Robotics' Bowler firmware was used to write namespaces and help communications between the embedded and computational systems. To complete the Bowler port, USB communication needed to be fully operational on the Abstract Hardware Device. Again, the LPC4330-Xplorer board was used to confirm this functionality. In order to test USB, the Lightweight USB Framework for AVRs (LUFA) had to link to the driver library and any other projects testing USB. This task proved to be more challenging than linking the example project from before, but eventually both the driver library and LUFA were linked and compiling. An echo server project was then created to test USB communications. An echo server receives a character from a CPU terminal via a serial communications port. The M4 core then receives the data and echoes the character back via the COM port after a brief pause. The serial communications port in this scenario was USB.

After creating an echo server project, properly linking that project to the driver library and LUFA, configuring the descriptor file to the Xplorer board, and downloading the project via JTAG, the echo server was ready to be tested. Initially, the echo server was tested in Microsoft Windows, but the CPU using that operating system was unable to establish a serial communications port with the Xplorer board. The reason that this test did not work in Windows was because of an issue with the driver

software being unable to access the board. After the failure of this test, the next logical step was to run the echo server in a different operating system. Linux Ubuntu was chosen to be the next operating system used because drivers are not necessary. By using a Linux operating system, the echo server was recognized through a terminal. This test proved that USB communication to the Xplorer board is possible in Linux but not in Windows. While the echo server functionality was never confirmed, the problem was determined to be with the descriptor file. WAVE's development can be continued by future teams, and in regards to the echo server, they will know to target the descriptor file in a Linux environment.

8.2.4.3 Embedded RPC Calls

During the development of the abstract hardware device, the ability to use the Bowler Protocol to send remote procedure calls had to be tested. This test was conducted using the Neuron Robotics' Dynamic Input/Output module (DyIO).The DyIO is able to interface with a CPU over USB and has multiple pin outs for peripherals. For the sake of testing RPCs, a potentiometer, a servo motor, and an LED were selected as the peripherals. All testing with the DyIO was conducted in a Linux Ubuntu virtual machine.

The first test was for the battery RPC. A function was written for getting battery voltage from any of the three batteries. Initially, three dummy values would be used to represent the batteries. After this functionality was confirmed, the potentiometer would be used to get values. The functions worked on their own with the dummy and potentiometer values; however, when trying to interface these functions with the Bowler RPC calls, the terminal did not display any values. Before trying to solve this problem, functions were written for the other two peripherals. This choice was made to determine if the problem was with the battery function or with interfacing functions in general. The servo motor was used for motor voltage RPCs. A function was written to send a PWM servo signal. Using the DyIO, this function was successful, but once again the function could not be interfaced with the Bowler RPCs.

At this point, the decision was made to use the LED to interface with already existing RPCs. The DyIO has a ping RPC, which flashes an LED on the module continuously when in use. An LED was configured to a peripheral port, and a function was written to mimic this functionality. As with the other two tests, the function worked independently, but did not work with an RPC. This third result was surprising because and already existing RPC was used. The problem was determined to be with the bridging loop between the Bowler protocol and all WAVE functions. When this problem was determined, not enough time existed to rectify this solution. Future teams will be able to see these results and know where to start with embedded RPC testing.

8.3 Software Testing

While a true test of WAVE's software framework would require a completed robotic platform to run on, it was possible to test the individual software components with simpler, stand-alone methods. These testing methods allowed for smaller segments of the code to be tested, as well as proving the functionality of the software without a fully operational robot. Several methods were used to test the functionality of the software, with unit tests and observation of various features running being the two primary methods.

8.3.1 Unit tests

Unit tests are a major part of any software development process. They ensure classes and methods exhibit expected behavior. They allow testing of specific components without running the whole program every single time. Java provides the JUnit framework designed specifically for testing purposes. These tests are a separate part of the main program – they have their own code, and are run independently. Testing every single part of the system would've taken too much time, so the focus was on testing the most critical robot components.

8.3.1.1 ObjectPipeEndpoint and the Poolside Interface

As the `ObjectPipeEndpoint` is a critical component in WAVE's operation, it required very careful testing. A series of JUnit tests were created to test the serialization, transmission, and deserialization of various objects. One such test case is included here:

```
@Test
public void testFirstObject() {
	// Send one Item over the pipe
	System.out.print("Adding Observer... ");
	client.addObserver(this, testObject);
	System.out.println("Done.");

	try {
		Thread.sleep(sleepTime);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}

	System.out.print("Sending First Object... ");
	ObjectServer.send(testObject);
	System.out.println("Done.");

	waitForObject();

	assertNotNull("Object not received over pipe.", obj);
	System.out.println("Comparing objects: ");
	System.out.println("\tOriginal: " + testObject.toString()); // Did you
	                                                          // get the
	                                                          // item?
	System.out.println("\tReceived: " + obj.toString());
	assertTrue("Object recieved of incorrect type.",
	obj.getClass().equals(testObject.getClass())); // Is it what you
	                                               // sent?
	assertTrue("Object received not equal to object sent.",
	             obj.equals(testObject)); // Is it what you sent?
	System.out.println("Success!");
}
```

This test method includes several logic checks which confirm or deny the successful transmission of an object over a local port. The test begins by observing the `ObjectPipeEndpoint` "client", for objects which match the type of the "testObject." The test then waits for a short amount of time to allow the operating system to handle the creation of the sockets, and for the connection to be secured. Then, the test object is sent to the `ObjectServer`. The test waits for a new object to be received, as

object serialization occurs in a separate thread. When `waitForObject()` returns, the new object is compared against the one sent through the ObjectPipeEndpoint. Several assert calls are used to determine its validity, including a null check to ensure than an object was received, an `assertEquals()` confirming that the new object has the correct type, and finally, an `assertTrue()` call which wraps a call to the test object's `equals()` method. With all of these tests returning successfully, the team was sure that the `ObjectPipeEndpint` functions correctly.

8.3.2 Observational Testing

Observational testing is simply testing different features and observing their operations. This is slightly different than unit testing. In unit tests, specific scenarios and cases are tested, and perhaps engineer failing cases on purpose to see if the program handles them correctly. Observational testing is just letting the feature run normally and noting any abnormal behavior. Several GUI components assist with this. For example, the log can be used to detect bugs with message outputs.

Observational testing is useful while testing multiple concurrent GUI connections. When a new GUI connects to the robot, the log displays information about the event. If nothing, or an error, is displayed, unwanted behavior is present. The same kind of logic can be used to test connections to the robot over the network as opposed to a local host. It can be observed from the log if the GUI connects to the robot, or if there was an issue. A simpler example is just running the GUI and seeing if the stopwatch runs correctly.

8.3.3 Memory Leak Testing

An important issue in software development is code aging. Aging refers to the progressive performance degradation or sudden hang/crash due to exhaustion of operating system resources. Memory leaks are a potential cause or contributing factor in software aging, as they can exhaust available system memory. To ensure no resources went to waste, Eclipse Memory Analyzer was utilized. This feature tracked how

much memory the code was using during operation. If system memory was not released after the software completed its run, this would indicate a memory leak.

Testing was done by leaving the code to run for some period of time and observing how it's using operating system resources. After completion of the test run, memory allocation was observed in an attempt to locate leaks or other issues. Using the detailed information from the analyzer allowed for optimization and bug fixes. This ensured correct resource allocation, no memory leaks and code durability.

8.3.4 Communications

One of the primary software deliverables of WAVE was to establish several communications pathways to facilitate the distribution of information throughout the system. In order to ensure the stability of these various systems, each one needed thorough testing.

8.3.4.1 Testing Bowler Communication

As stated previously, one of the key factors when selecting the Bowler Communications System was the availability of the DyIO to test preliminary functionality. The DyIO, a small embedded microcontroller developed by Neuron Robotics is tightly integrated with the Bowler Communications System out of the box, and supports a wide range of functionality. DyIOs were used to test various aspects of the software system, including the generation and interpretation of Bowler Command Packets. The team was able to simultaneously test the mission control system and Bowler communication through the use of simple blink pattern mission files. One such file, included here, synchronously blinks the SOS pattern through an LED, while asynchronously blinking another LED at a fixed rate.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Mission name="Test Mission 2">

	<Task type="Echo" message="Commencing SOS!"/>

	<Task type="AsyncBlink">
		<Device>BlinkTest</Device>
		<Port>12</Port>
		<Duration>200</Duration>
		<Gap>200</Gap>
		<Inverted>False</Inverted>
	</Task>

	<Task type="Blink">
		<Device>BlinkTest</Device>
		<Port>3</Port>
		<Duration>300</Duration>
		<Gap>150</Gap>
		<Count>3</Count>
		<Inverted>False</Inverted>
	</Task>

	<Task type="Wait">
		<Duration>500</Duration>
	</Task>

	<Task type="Blink">
		<Device>BlinkTest</Device>
		<Port>3</Port>
		<Duration>600</Duration>
		<Gap>150</Gap>
		<Count>3</Count>
		<Inverted>False</Inverted>
	</Task>

	<Task type="Wait">
		<Duration>500</Duration>
	</Task>

	<Task type="Blink">
		<Device>BlinkTest</Device>
		<Port>3</Port>
		<Duration>300</Duration>
		<Gap>150</Gap>
		<Count>3</Count>
		<Inverted>False</Inverted>
	</Task>

	<Task type="Wait">
		<Duration>1200</Duration>
	</Task>

</Mission>
```

This mission begins by logging a message, stating that it is about to send the SOS pattern. It then spawns a new `AsynchronousTask`, running in a new thread, which will periodically blink an LED attached to port 12. The duration and frequency of these blinks is specified within the XML elements included within the `AsyncBlink` task element. It then begins a series of timed blink messages, which control a separate LED on the attached DyIO. As you can see from the XML above, the durations of these blinks varied, creating the different Morse code symbols.

8.3.4.2 Testing AHD Communication

Unfortunately, the Abstract Hardware device was not downloading code correctly until the last days of the project, and has yet to run the Bowler Server. However, as soon as the Bowler Server is downloaded, and all of its drivers written and functioning, the DyIO from the team's previous tests should be easily replaced with the AHD. As the AHD completes several checks to ensure that its core functionality is stable (such as the SOS blinking test mentioned above), the Electrical team can begin to implement WAVE's custom RPC calls.

As with the initial testing of Bowler Command packets, the functionality of these custom RPC calls can only be verified through various methods of observation. For example, to test the motor speed RPC call, you could measure the PWM channel of the AHD with an oscilloscope. As different speeds are commanded by subsequent motor velocity RPC calls, the PWM signal should change accordingly. This could then be tested outside the robot by attaching the motor driver board, and some extra motors. By observing their rotation and speed the functionality of the motor velocity RPC call could be verified.

8.3.5 Poolside Interface Communication Testing

During the construction of the GUI, the team noticed a particular exception, which were thrown periodically by the `ObjectPipeEndpoint`. This `ConcurrentModificationException`, seemed to plague our systems, as it was thrown at irregular intervals, and was not readily reproducible.

This error was not catchable by the `ObjectPipeEndpoint`, and could not be handled dynamically. After extensive debugging, it was determined that this error was caused by concurrent access to the shared `RobotModel` objects.

To control access to these objects, a simple mutual exclusion lock was created to wrap these models. The `RobotGlobalModel` class is shown below, showing the locks and the objects they contain.

```
public class RobotGlobalModels {
	private static LockedObject<RobotModel> robotModel =
		new LockedObject<RobotModel>(new RobotModel());
	private static LockedObject<MissionModel> missionModel =
		new LockedObject<MissionModel>(new MissionModel());
	private static LockedObject<LogArchive> logArchive =
		new LockedObject<LogArchive>(new LogArchive());
	private static TaskManager manager = new TaskManager(missionModel);
	private static SystemClock clock = new SystemClock();

	static {
		robotModel.getObject().setCurrentLocation(
			new TransformMatrix(0, 0, 0, 0, 0, 0));
		robotModel.getObject().setDesiredOrientation(
			new TransformMatrix(0, 0, 0, 0, 0, 0));
	. . .
```

The file continues, initializing the remaining models, and implementing several simple getters and setters, which have been omitted here for brevity. These `LockedObject` objects extend Java's `ReentrantLock` class. This standard Java element provides the functionality of an atomic lock, synchronizing access. The `LockedObject` class implements a method `getObjectWithLock()` which returns the object which it wraps. If this lock is currently held by another thread, this method will block until the object is unlocked. By surrounding access and modification of these objects, proper concurrency was restored, and the `ConcurrentModificationException` stopped being thrown.

8.4 Integrated System Testing

While the team was never able to do fully integrated system tests due to a number of technical problems which prevented WAVE from becoming fully functional, a number of tests were planned. If WAVE had been fully integrated into a functioning system the following tests would have been done to prove the system's capabilities.

8.4.1 Fully Integrated Submersion

The first planned test was a fully integrated submersion. This test comprised of two parts. Firstly, the integrated system would be submersed with all of the electronics offline. This procedure allows us to test the electronics housing as it supports the weight of all electrical components. Also, removing power from the electronics better ensures their safety, should this test reveal a new leak. Once the first submersion test was completed, the next step would be to run a test with all of the electronics online. This second part would validate the functionality of the electronics while WAVE is submerged. This test is for submersion purposes only and navigation would not be executed.

8.4.2 Safety System Tests

In order to ensure WAVE responded appropriately in case of an emergency, two safety tests would be conducted. The first test would be to activate the E-Stop switch from the poolside user interface. Upon activating the E-Stop switch, WAVE's power system would deactivate. This system was planned but not designed in this version of WAVE. All of the electronics would be disabled and prevent damages from occurring. Secondly, another test would be conducted to ensure that the emergency ascension protocol functioned. WAVE would sense internally that the platform was in a dangerous situation and trigger the process of surfacing.

8.4.3 Maintaining Position

At certain points during missions, WAVE would need to remain in place. This capability would be tested through a combination of thruster control and active ballast. The thrusters and ballast would work in tandem to allow WAVE to 'hover' in place.

8.4.4 One-Dimensional Motion

With six degrees of freedom, WAVE's one-dimensional movement would need to be verified. Movement among the lateral x and y axes would be tested through the thrusters. Additionally, the thrusters would work with the ballast system to control depth and confirm movement along the z-axis. Once lateral motion was validated, rotational movement would be tested. The thrusters would be tested in their ability to generate roll and yaw, while the ballast system would be used to generate changes in pitch.

8.4.5 Two-Dimensional Motion

Upon the verification of one-dimensional motion, movement throughout two dimensions would be tested. Each combination of movement described in Section 0 would be tested. The completion of these tests would validate WAVE's ability to successfully maneuver through any underwater environment. This process requires extensive future work in order to achieve functionality.

9 Future Work

Several additional improvements could be implemented by future teams to enhance WAVE. The basic platform could be further developed, a number of new modules to augment WAVE's abilities could be created, and all of this could come together to make WAVE ready to compete at the AUVSI Robosub Competition.

9.1 Platform Improvements

WAVE is still in the fairly early stages of development. In comparison to the long-term sustainability of WAVE as a research platform, this first year of development is only a fraction of the capabilities. Many platform improvements can be incorporated in the mechanical, electrical, and software systems to increase the scope of WAVE.

9.1.1 Mechanical Improvements

There are several improvements that could be added to the mechanical subsystems on WAVE. The chassis could be redesigned to allow for more modular mounting points with the addition of more 80/20 aluminum crossbars. The electronics housing could be redesigned to allow for separate chambers for batteries. These chambers would provide a barrier, preventing the batteries from affecting the other components in the electronics housing if a failure were to occur. Additional connection ports would be beneficial by allowing a greater number, and thus a wider variety, of modules to be connected to WAVE. One of the project's goals is to create a design that is customizable to the needs of the user, and more connections would give latitude in that regard. However, there is a limit to expanding this connectivity, as more connections are also more potential leaks and eventually the risks would outweigh the benefits of continuing to add connections.

9.1.1.1 Electronics Housing

The electronics housing could be improved. Currently the electronics in the housing do fit, but the size restrictions limit the WAVE platform to two batteries. The size of the housing could be increased to allow for an additional battery as well as for easier wire placement within the housing. Additional measures could be taken to ensure adequate thermal dissipation such as the addition of thermal paste or conductive materials to the walls of the housing. More connectors could be added to the housing to allow more modules and components to be added to the WAVE platform. Additionally, the batteries could be moved to a separate housing for easier access and an external power switch could be added to reduce the need to constantly access the electronics housing.

9.1.1.2 Frame

The frame's overall size could be increased to allow more room for the wiring and tubing around the electronics housing. A larger frame could also benefit the platform by allowing modules to be shielded within the dimensions of the frame's edge, reducing risk of damage to modules and components during use of the platform. However, to stay within the boundaries established by AUVSI, the frame can only be increased by 49.7cm in with, 72.6cm in height, and 111.3cm in length. Further, manipulators or extendable modules could be designed to fold into the area inside the frame, keeping them protected as well. Work could also be done to reduce the weight of the frame such as with the Killick Frame, which is discussed in Section 9.2.3. This would reduce the load on the ballast system and overall increase the agility of the platform.

9.1.1.3 Ballast

Regarding the ballast system, there is much that can be done to improve the overall design. Adding sensors in the tanks to measure the pressure, or flow meters at the nozzles would allow greater control, and give the robot the ability to fine-tune its vertical position in the water. The tanks could be redesigned to have a more accurate fit to the nozzles being used to connect the tubing from the pumps to the tanks. The current nozzles need to be reinforced with epoxy resin to prevent leaks. The tanks

could also be remade out of acrylic or another transparent material to allow for more accurate observation and testing. It is currently impossible to directly observe the water in the tanks.

While the motors being used to power the ballast system work well for the project's needs, they are not ideal. They could be replaced by waterproof motors which would have a longer performance life while working underwater. Further, the pumps being used on the robot, while great as an initial prototype, place an incredibly high torque requirement on the motors being used. The required torque to start the pump is approximately 1.1Nm (9.5 in-lbs), a number which many motors within the acceptable weight and size range that are usable on the robot have trouble meeting. These could be replaced with different positive-displacement pumps that require less power to turn.

The attachment method used for the ballast system, specifically it being located on the side of the robot where the electronics housing will be opened, can be altered to allow easier access. As it stands, the user must unscrew the cable ties to remove the tank to gain access to the opening of the electronics housing. These cable ties could be replaced with a latch system that would be undone with minimal effort. An alternative design using a hinge system could allow users to swing the tank out of the way so that it never has to be truly removed from the frame. Lastly, an external kill switch must be implemented in case of an emergency.

9.1.1.4 Thrusters

While the motors on the submarine are serviceable, they leave room for improvement. More efficient motors could easily be fitted to give WAVE a longer run-time. These motors would drastically reduce travel time while underwater, allowing for the collection of greater quantities of data and more time to complete objectives set by future WAVE users.

9.1.2 Electrical Improvements

In addition to mechanical changes, future teams will be able to expand upon WAVE's electrical infrastructure. The extensibility of WAVE's electronics is evident, and teams can extend the modularity with future development and improvements.

9.1.2.1 USB Host Functionality

USB device capabilities were examined in the design and analysis, but host functionality was not taken into consideration for this iteration of WAVE. Specifically, the AHDs are currently designed to use USB to communicate with the fit-PC as a device. None of the components of the sensor suite communicate via USB. This decision was reasonable for the initial development of WAVE, but in order to account for future growth as a research platform, host functionality should be considered. AHDs as hosts would be able function as a central processing unit, similar to the fit-PC. The inclusion of this capability will increase the utility of the AHDs in the system. The number of sensors and modules that would be able to interface with WAVE would increase due to the ability of these components to be USB devices. Finally, the processing load on the fit-PC will be reduced, as the CPU will no longer be the only USB Host on WAVE. Proficiency in USB devices (on the part of the development team) will be necessary as this standard can be difficult to implement. Future teams must plan to allocate a sufficient amount of time to achieve this functionality if they decide to include it.

9.1.2.2 Boot loader

Currently, the only way to download code to the AHD's core is via JTAG communications. This method is acceptable for debugging code; however, using the JTAG can be tedious and waste copious amounts of time. Since the AHDs will already be connected to the Fit-PC via USB, implementing a boot loader would provide an additional way to download code. Using a boot loader would allow for the run-time environment to be sent from the Fit-PC to the AHDs after the completion of internal self-testing. This

capability enables programming on-the-go, which would be a huge upgrade towards making WAVE modular.

9.1.2.3 Sensor Upgrades

For this system sensors were implemented to measure the following parameters: temperature, humidity, water leakage, and pressure. These are the bare essentials of what is needed in order to implement a working system. For the future, the WAVE modular system can be improved with implementation of more and perhaps additional sensors. One specific example is that flood sensors should be placed in more than two locations in the pressure vessel in order to ensure that leaks can be rapidly detected regardless of the current attitude of the WAVE system.

9.1.3 Software Improvements

While WAVE's current software suite provides a strong foundation for running WAVE's tasks and delivering important information to users, there is still room for improvement.

9.1.3.1 Integration with AHDs

Due to the troubles with getting embedded software running on the AHDs and control boards, limited work was done for PID control functionality. This would be the highest priority to make WAVE's software more useful for actively driving the robot. Fully implemented PID control would allow WAVE to properly take sensor data into account and properly adjust motor speeds and directions both for maintaining position in the water as well as successfully traveling between waypoints in its environment.

Additionally, more work could be done to relay additional sensor data to the user of the Poolside interface. As sensors are added, whether these are cameras, pressure sensors, or something else entirely, the ability to see what these sensors are currently reading in the GUI could be of immense use for debugging purposes.

9.1.3.2 Poolside Interface

The GUI itself has room for improvement as well. The attitude indicator could be made smaller, allowing room for other indicators and/or gauges. Luckily, the JFreeChart library has already been included with the latest copy of the code, so the addition of several gauges would be rather trivial. Mission controls could be implemented as well, such as stop/pause task, go to next/previous task, restart task/mission. To help with mission control, a map showing current, starting and goal position, as well as speed and heading, could be implemented. A status bar could show estimated time of mission completion, as well as how many other GUIs are currently connected to the robot. There are many ways that users could use the GUI to interact with WAVE in the future, but these implementations would depend on the needs of different groups and their missions.

9.1.3.3 Software Testing and Validation

In terms of testing the code, there are several more opportunities for testing that were no pursued by the team. Test driven development was considered, but was deemed too costly (as it required tests be written first, then the code to make the test successful) and not really applicable to WAVE's design. The software either required the physical devices or had to be "tricked" into thinking that it had those devices in order to test code. Because of limitations such as these, the team fell behind on writing test cases. Instead, several parts were fully developed by going back and forth between writing code and just running it to see if it works, which is not proper software testing procedure. Improvements could include incremental testing of the various code features, instead of waiting to have major components done before starting to test them. Thorough testing of several of the utility functions (such as the transform matrix functions) could have been done. Several other aspects, such as boundary conditions and robot models, had little to no testing done before deployment. Major components, such as these, should have received more testing to confirm desired functionality.

9.1.3.4 Mission Enhancements

Finally, in conjunction with functional embedded code, task management functionality could be greatly expanded. With fully functional embedded software managing WAVE's systems, more RPCs could be implemented to send data between the Fit-PC and the control boards, and tasks could be added to handle these additional inputs and outputs and do various actions based on them. Tasks could be made to be paused or handled in a different order based on what WAVE perceives in its environment, as well as have tasks be able to generate other tasks as necessary. For example, if a locomotion task is driving WAVE to a point, but along the way the sensors detect an item to retrieve, WAVE could react properly. WAVE's task system has the capability to be extended so that in this example, WAVE could pause the current locomotion task, generate a task to retrieve the item with an arm, and then continue onwards to the destination. Furthermore, the addition of basic flow control logic, such as if/else, while, for loops would allow for much greater complexity within a mission

Similarly, a simple messaging system should be established for GUI communication. This would allow messages to be sent to a particular "Master" GUI, which will block until cleared by the user. The robot would then

9.2 Future Modules and Features

Given more time and money, modules and features would be added to the device. The following modules are of interest for future work: a mechanical arm, a torpedo launcher, active ballast, and improved motors. Both the mechanical arm and the torpedo launcher would be necessary for the AUVSI competition. Reduced weight would be a good feature to include in future iterations of this design. Additionally, bio-inspired propulsion would be an interesting field in which to perform research and development.

9.2.1 Future Module Development

As stated in the sections about the abstract hardware device the AHD serves as the standardized embedded computing platform on which WAVE, WAVE's modularity, and hardware expansive nature is based upon. Using the AHD and the AHD Shield Hardware Development Kit (HDK) an additional module performing any AUV centric function can be rapidly developed and implemented for WAVE. The AHD Shield HDK is an Altium Designer project template that includes a schematic document, part library, and a PCB document. In essence, the AHD Shield HDK is a premade Altium project for making shields for the AHD. This HDK reduces the difficulties associated with designing a shield, by default the PCB shape is premade and locked, all the headers are preplaced and locked on the PCB, and a complete library of all the components for the shield are included. Given the setup provided in the HDK all a designer needs to do is add the circuitry of their module to the premade schematic and import the components into the PCB document and lastly run the traces. By providing a comprehensive HDK for module development using AHDs the hope is that many future modules will be design and implemented on WAVE.

9.2.2 External Modules

An arm could be added to facilitate the manipulation of elements around the submarine. As a testing platform, this design, as well as a successful implementation of an arm, would be invaluable to study projects. In the shorter term, many AUVSI challenges require the ability to actively manipulate external objects to achieve a goal.

Torpedo tubes or more specifically, launch bays would be useful in the deployment of sensor arrays. The ability to load and deploy different types of payloads such as sensor clusters for data gathering would give future groups a chance to more easily gather data from underwater research.

9.2.3 Weight Reduction

In addition to more modules, additional features could also be improved on the device. Reducing the weight of the robot could reduce the overall expense of its transportation. A smaller staging crew and transportation vehicle could be utilized to get the WAVE platform to its mission area. Because the focus is to build an accessible and easy to use base platform, ease of transportation would greatly increase the viability of the design over a wider spectrum of groups.

9.2.4 Bio-Inspired Propulsion

Research could also be done into the field of bio-inspired propulsion, such as fins and sea-life appendages. Many sea creatures possess efficient modes of travel, and it could benefit the design to consider such alternatives. Jet propulsion as inspired by a squid was considered initially. Also fish fins could be explored. A robotic fish is shown in Figure 93. Gills were designed that allow this robot to move freely through the water. In relation to the current design, the thrusters could be replaced with fins or other propulsion systems inspired by marine life. This substitution would not necessitate a complete redesign of the WAVE platform.

Figure 93: Robotic Fish

9.2.5 Additional Actuator Control

Further actuator control could be implemented in future iterations of WAVE. This additional control would extend beyond the scope of thrusters. AUVSI competitions and certain missions could require an external manipulator to complete tasks. Any manipulator added to WAVE would require an actuator board. These boards would be of similar structure to the thruster and ballast boards and would communicate with AHDs. Additionally, any manipulator would require custom RPC calls specific to the module. Finally, additional actuators would need to be integrated with the waterproof connectors on the end-cap. Extra connection points are available on the end-cap for actuators that can be added to WAVE's modular system.

9.2.6 Robot Simulation Suite

A common method for testing robots without having to actually run them in their environment is to use software simulations. However, when the team investigated the availability of free or low-cost robot simulations, none were discovered for underwater robots. A software simulation environment would allow future WAVE teams to simulate how WAVE would perform under varied conditions and load outs. This allows testing outside of the pool, saving significant time, effort, and, in case of unexpected failures, money.

This suite could include various features to aid teams in their simulations. To simulate an underwater environment, there could be an option to add obstacles, such as rocks, fish, and buoys. One could also configure the depth, pressure, and temperature of the water, as well as simulate water currents. The simulation run could show current position, speed and direction, with the ability to move to checkpoints of a predefined route. The interface of this suite could have the option to add sensor data noise for a more realistic simulation. Other aspects of this interface could include the ability to arrange and configure the robot modules in order to determine weight and expected power usage.

Based on that configuration, one could select model and power for thrusters. Additionally, actual battery usage based on current configuration and activity could be displayed. Another feature of this suite could be the ability to upload the actual robot JAR file used to run WAVE. Then, the suite could simulate an environment for the robot, such as generating artificial sensor data. This way, the uploaded code could be tested if it's working as expected.

9.2.7 Modular and Customizable User Interface

When it was decided that WAVE's user interface should allow for multiple connected clients, one of the goals was to allow for different members of the WAVE team to focus on viewing different information simultaneously. However, in its current form WAVE's GUI displays the same information to every user, which makes having multiple clients less necessary. As WAVE's functionality expands, more features will likely be added to the GUI, which could potentially lead to a very cluttered and difficult to manage interface. This could be avoided by making the GUI modular and customizable.

Making a modular and customizable GUI would mean allowing for the GUI to be divided into different "modules" some of which could correspond to physical modules attached to WAVE, and others to more general objects such as the log. Users could change out different GUI modules at will depending on what information they wanted to view, and could potentially change the size and location of such modules on the screen to allow for a more customized and personal UI. This process could also be automated, with WAVE potentially having a number of default configurations that could depend on what modules and functionality WAVE currently has, which can already be easily determined using the existing properties and devices files. An additional possibility would be to have separate GUI configuration files that different users could customize and use on their personal computers. The software could then find and access these files on startup, allowing each individual user to potentially

start up the GUI with a different configuration. These options, while not necessary to WAVE's performance, all could contribute to a more useful and intuitive interface for WAVE's users.

9.3 Requirements for AUVSI

In order to have a chance of being successful in the AUVSI competition, the robot requires several systems that are not currently included in its design. A vision processing unit capable of color recognition is required for the different types of challenges that AUVSI poses to the competing teams, such as this year's stoplight challenge where the robot needs to be able to recognize the color a stoplight is showing and respond accordingly. A recurring theme in the AUVSI competition is the use of a torpedo launcher to score points by shooting a torpedo through different sized holes in a wall. As such, the robot will need a torpedo launcher for this portion of the challenge. Finally, a manipulator will need to be added that can handle different course objects. This year, the objects in question were a steering wheel and a shifter, simulating the driving of a manual transmission car, and an object that simulates delivery of a box of food. These are the basic requirements that are needed to be successful in the range of challenges AUVSI poses to teams each year, and having modules to meet these needs would greatly increase the viability of WAVE.

10 Conclusions

Upon the completion of this project, multiple conclusions were drawn. Certain tactics were deemed as being successful for the growth of WAVE, whereas other tactics needed to be reevaluated and reconsidered. In regards to the tactics that were unsuccessful, possible changes to their approach were determined. While a lot of time was put into determining different approaches, the project also yielded a multitude of results. These accomplishments will help future teams further understand how to work with and continue to develop WAVE as a research platform.

10.1 Successful Tactics

Tactics labeled as successful were ones that had a major positive impact on the project experience. These tactics were generally more thought-out and serve as a model for the thought process that should be put into every idea.

10.1.1 Full-team Meetings

For the majority of the year, the individual sub-teams met on a regular basis; however, the only regularly scheduled inter-team communication was the general body meetings. During the Spring Break, the entire team met every day, and these meeting yielded positive results and progress. After the break concluded, the whole team continued to meet regularly. At these meetings, the sub-teams were able to deliver progress and get consistent feedback and suggestions from the other groups. While meeting at the integrated team level did not seem overly important at first, this belief was proven to be incorrect. Strong inter-team communications leads to more consistent results, and full-team meetings were a strong contributor for WAVE's accomplishments.

10.1.2 High Level Software Development Environment

Choosing Eclipse for the high level software development environment proved to be an excellent choice. Due to the familiarity from robotics and computer science courses, Eclipse proved to be a reliable choice

with many helpful resources available on campus. Additionally, the use of the Bowler Protocol as the bridge between high-level and embedded computing was also helpful. The ability to speak with the firmware's developer (Neuron Robotics) on a regular basis proved critical. This decision shows that using reliable sources is more important than potentially more powerful ones.

10.1.3 Sponsorships

The team's budget was relatively small for the number of participants in the project as well as the scope and complexity of WAVE. From the start, the ability to get parts donated was deemed critical. While a formal sponsorship sub-team never came to fruition, many individuals were able to obtain sponsors. Many parts were obtained in exchange for promoting the sponsoring companies on the team website, poster, and robot. These donations saved the team thousands of dollars. Without sponsorships, these parts would be unattainable with the given budget. Rather than compromise the design process, having sponsorships enabled the team to use optimal components for many of WAVE's subsystems.

10.1.4 Team Website

Having a team website was a successful tactic for obtaining sponsors and donations. The website was updated on weekly basis with blogs. These blogs outlined the progress that the three sub-teams had made in the previous week and events for the upcoming weeks. Team bios existed and provided a brief description of each member and advisor. The website had the ability to upload media. Pictures and videos of WAVE's progress were provided for site visitors. The level of professionalism about the website showed visitors the magnitude of WAVE and encouraged potential sponsors to provide donations to the team.

10.1.5 Microsoft SharePoint

Using Microsoft SharePoint enabled the team to exchange documents and information through a reliable source. Many documents including this report could be modified by multiple team members at

the same time when using SharePoint. Wiki pages were used to store meeting minutes and agendas. The picture library stored vital design images and flowcharts. In the event that an unintended change was made to a file, SharePoint's version history allowed the team to revert back to previous revisions with relative ease. Lastly, SharePoint documents when people upload or make changes to files. This accountability helped ensure that each team member would work reliably.

10.2 Reconsiderations

Not all of the team's approaches were one hundred percent successful. Many of these tactics needed to be reconsidered throughout the project. Unfortunately, most of them hindered the development of WAVE, but they serve as a warning for future teams.

10.2.1 Organization

From the start of the project, a lack of organization existed. As stipulated earlier, the team did not meet as a whole very often. Even a lack of communication within the sub teams existed for a period of time. Unfortunately, this disorganization hindered the progress of the project at times. Miscommunications caused problems throughout the year. While senior-engineers were elected each term to serve as organizational leaders of the sub-teams, this approach proved to be flawed. The senior-engineers had to divide their time between technical and organizational aspects of the project, and as a result both parts faltered. In hindsight, the team could have employed a project architect who would be in charge of organization and responsible for resolving technical disputes. A project architect would have streamlined many of the team's major technical decisions, while designing to a fixed set of specifications. As any difficulties arise, the project architect would be tasked with devising a solution instead of having all the sub-teams struggle to reach a consensus. With a project architect and engineers putting their full efforts towards their prospective tasks, WAVE could have enjoyed more success.

10.2.2 Scope

One problem encountered with such a large team was an overestimation of the project's scope. The strong inter-team dependencies caused large decisions to be decided upon in an untimely manner. Because of this, a lot of the team's goals were unable to be accomplished. With such a large scope, problems inevitably arise. The team underestimated the amount of time needed to complete certain tasks. For example, creating WAVE's platform from scratch hindered the fully integrated systems testing. If the scope of the project had been to create a platform, the goal would have been more easily obtained. Even though the team understood that certain goals would not be reached, the expectation was still to have a fully functioning platform by the end of the year. The combination of high expectations and a large scope set the team up for disappointment at different times. With a more reasonable scope, WAVE would have more completed features. Future teams should take this fact into account when planning to work on WAVE.

10.2.3 Documentation

Many may not see this problem at this point in time, but the documentation throughout the early stages of WAVE's development was not particularly strong. With the teams meeting on an infrequent basis, questions could have been answered better through online documentation. The team had a Microsoft SharePoint page. While SharePoint was used to share important documents, this communications tool was not utilized strongly at first. Rather than keep important decisions documented, the sub-teams would schedule meetings that sometimes took weeks to happen and wasted valuable time. When this process was deemed to be ineffectual, the team began to utilize the wiki-pages on the SharePoint to store meeting minutes. Whenever a sub-team had an important question, they could reference the notes before scheduling an unnecessary meeting. This reevaluation allowed for the team to utilize time better for WAVE, but the time lost in the beginning of the year would have to be made up for. Documentation will be important for future WAVE teams in order to better utilize their time.

10.2.4 Unfamiliar Embedded Environment

While LPCXpresso is a highly touted software development tool, the decision to use this unfamiliar embedded environment was, in retrospect, not a wise one. The major advantage of using Eclipse as the high-level software environment was familiarity with the software. WPI utilizes Eclipse for all robotics classes as well as numerous programming classes. Therefore, if a problem appeared, the team would have multiple resources to help fix the situation. With LPCXpresso, no one was available to help resolve problems. This lack of help, coupled with inexperience with the software, led to many problems. Code-Red's technical support was not very helpful in resolving these issues as well. Due to the choice to use the free software over the paid versions, the project was not deemed as a priority by Code-Red. If the team had chosen software and a development environment that was more familiar, fewer problems would likely have appeared, and people and resources would be available to help solve any difficult problems.

10.2.5 Funding

Sponsorships were a key component to aiding the team's budget issues; however, not enough people obtained outside funding. A few individuals got sponsors, which were huge developments, but everyone should have been actively looking to get funding. The team had to buy many components out of pocket. Due to the lack of funds between the team members, the optimal components could not be purchased in every scenario. With a more active pursuit of outside funding, WAVE's infrastructure would be comprised of the most optimal components.

10.2.6 Budget Management

Although the small amount of funding was a problem, the bigger problem was budget management. Even though the project consisted of three sub-teams, the budget was allocated for the entire group. As a result, reckless spending ensued. Sub-teams ended up spending more money than was allocated for them, and the team ended up in debt to the robotics department. Shortly after, a budget manager was

appointed, and the budget was balanced. Having a budget manager was a great change, and this reassessment allowed the team to make purchases more carefully. Future WAVE teams should appoint someone to this position from the beginning of their project.

10.3 Possible Changes

As with any project, the team noticed opportunities that weren't be maximized. Possible changes to procedures would allow for optimal efficiency. All of these changes were briefly touched upon in the reconsideration section. These possibilities should be strongly evaluated by teams wishing to continue the development and use of WAVE.

10.3.1 Stronger Organization

The team felt that the organization of the project was a problem. The individual sub-teams were organized, but there were inconsistencies at the group level. Stronger organization should be an important factor in the future. The project team needs to meet on a regular basis. Documentation must also be better. It would be a good idea to employ a management major on the team to be in charge of all organization. Additionally, budget management must be taken into consideration at all times. Basically, every aspect of the project must be more organized. In the future, teams should plan the organization process in the year before undertaking the project. Working on WAVE requires organization at all times, and teams cannot afford to fall behind at any point in the project.

10.3.2 More Communication

Communication is extremely important with a large team but was not properly executed throughout this project. Firstly, sub-teams need better communication with each other. Many design decisions had to be postponed because of uncertainties with the other teams' specifications and abilities. Communication on a regular basis would solve this issue. The communication between students and advisors could have also been better. Certain requirements for the project were ignored for a long time because no one was

asking questions about them. Assumptions were made that some criteria might not be necessary without any confirmation. WAVE teams should make sure to communicate with their advisors beyond the general body meetings and not be afraid to ask questions about any uncertainties.

10.3.3 Strongly Enforced Deadlines

Many deadlines were set throughout the year, but they were not strongly enforced. Frequently, teams would miss a deadline, and WAVE's progress was stifled. Deadlines did not appear to be important from all sides of the project giving people no incentive to complete them on time. Managers need to be able to keep people accountable for their deadlines. A suggestion for future teams is to develop a system where individual progress is measured as well as team progress. A Gantt chart was discussed but never implemented. Such a chart would serve as physical evidence of an individual's contributions and setbacks, and everyone would be held more accountable.

10.4 Accomplishments

This project achieved many technical goals in the first year of WAVE's implementation. These accomplishments go a long way in showing proof of concept for WAVE's design and analysis. Future teams will be able to observe these achievements and obtain a better understanding of what they can accomplish with WAVE.

10.4.1 Chassis

The frame was successfully assembled in the configuration necessary to seat the electronics housing and to allow the motors and ballast tank to be properly attached. The 80/20 brand aluminum gives latitude for the placement of the components and later modules. The electronics housing was machined to specifications and the connectors were attached to the front plate. Fittings were machined to attach the electronics housing to the frame.

10.4.2 Software Framework

The software team was successful in utilizing the NRSDK. The most relevant aspects of this framework were incorporated into the code design. More specifically, the AHD design and several task classes are the objects that implement the NRSDK. The main aspects utilized from this framework were the DyIO controls, serial connections, channel controls and representation of a Bowler abstract device.

10.4.2.1 Communications

The Bowler Communications Protocol was the major component in the development of communications for WAVE. The software team found success in developing several serial methods of exchanging data. Custom RPCs can be developed for the appropriate robot tasks and then uploaded to a DyIO. The RPCs can then be tested by creating an appropriate task for that RPC. Additionally, communication between the IMU and the GUI has also been developed, with the GUI receiving the appropriate data.

The software team was successful in developing an `ObjectServer`. The server was responsible for accepting GUI connections. With each GUI, the server spawns a thread. Upon a new connection, an `ObjectPipeEndpoint` object is spawned to handle communications between the GUI and the robot. As each GUI starts, it initiates a connection to be accepted by the `ObjectServer`. The new GUI is accepted by the server and it passes responsibility for all further connections to the `ObjectPipeEndpoint`.

The Model View Controller design pattern was successfully utilized as well. In WAVE's case, the view is the GUI, and the controllers are the environment in which the robot operates. Sensors detect any changes and update the appropriate models. These updated models are then sent to all connected GUIs and the view is updated. This MVC pattern is slightly modified to include a network socket. This socket serves as a "gate" through which all models are transmitted to the GUIs. The

`ObjectPipeEndpoints` are responsible for serializing the given models, transmitting them across the network, and de-serializing the objects when they arrive at the GUI.

A new class, `TypeObservable`, was designed in order to facilitate the distribution of new objects from an `ObjectPipeEndpoint`. When a `TypeObserver` registers with a `TypeObservable` object, they include an example object of the type of object they would like to observe for. The `TypeObserver` pattern enables the `ObjectPipeEndpoint` to distribute new objects as they are sent to the various poolside interfaces. While these endpoints are bi-directional, it does not enable the GUIs to command the robot in any way.

10.4.2.2 GUI

The GUI accomplished several major things. WAVE's poolside interface is successfully able to communicate with the robot. The GUI can receive mission progress, system uptime and log messages. All of the components update based on the robot status. The attitude indicator updates based on data received from the IMU. The mission tasks update as the robot progresses through them. All messages are sent to the log, even if they're sent at the same time or from different threads. Log filters are implemented, and filter messages based on their type, such as error or task status. System uptime starts as soon as the main robot code is started, and then successfully synchronizes between GUIs, depending on the time that they connected to the robot. The emergency stop button correctly creates an E-Stop event and is put into the log.

10.4.2.3 Task Management

WAVE's software system is successfully able to manage a series of tasks as part of a larger mission. These missions are fully customizable by users and can be easily swapped allowing WAVE to change its functionality without having to modify and recompile code. WAVE is able to take a list of synchronous and asynchronous tasks, and successfully iterate through them. Asynchronous tasks are started in their own threads while synchronous tasks are iterated through in order. Creating new types of tasks is simply

a matter of creating a new Java class that extends the abstract task class and implementing whatever functionality is needed, allowing for easy expandability to handle more varied and complex tasks in the future as WAVE's capabilities grow.

10.4.3 Electronics Housing

The electronics housing is a pressure vessel fabricated out of extruded aluminum rectangular tubing with end-caps attached on either end. The housing was designed and fabricated keeping waterproofing and thermal considerations in the forefront. Each end-cap had a groove machined, which was then filled with a silicone gasket to prevent leaks. The housing contains an electronics rack, which is used to support electrical components of WAVE, including the fit-PC, several sensors to monitor internal conditions, and all of the PCBs. Several waterproof connectors go through the end-caps in order to connect the thrusters, ballast system, and any external sensors.

10.4.4 Power System

The power system achieved high modularity and is capable of powering most any mission that WAVE could embark on. Printed circuit boards capable of sourcing high power were successfully created. With this, capability to distribute power throughout WAVE's entire system was achieved. High powered lithium polymer batteries were obtained and familiarity was gained with this newer battery chemistry. The requirement of obtaining a run-time of at least 20 minutes was fulfilled. Voltage rails of 18.5V, 12V, 5V, and 3.3V were successfully created and are available throughout the system providing flexibility when choosing components for future modules.

10.4.5 Sensor Suite

A sensor suite was designed incorporating sensors capable of detecting temperature, humidity, and compartmental flooding. An AHRS/IMU was chosen and implemented to show working functionality

with the GUI. Also a pressure sensor was chosen in order to measure depth. The internal sensor suite was breadboard tested and proved working functionality as an integrated subsystem.

References

[1] SINTEF, "Robots Taking Over the Job On Offshore Oil Drilling Platforms," 1 January 2008. [Online]. Available: http://www.sciencedaily.com/releases/2007/12/071221230852.htm. [Accessed 10 December 2012].

[2] "Undersea Robots for Deep Sea Exploration," [Online]. Available: http://www.me.jhu.edu/r_ocean.html. [Accessed 10 December 2012].

[3] G. M. Trimble, "Autonomous operation of the Explosive Ordnance Disposal Robotic Work Package using CETUS untethered underwater vehicle," [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=532396&tag=1. [Accessed 10 December 2012].

[4] "Bluefin-12S," [Online]. Available: http://www.bluefinrobotics.com/products/bluefin-12s/. [Accessed 10 December 2012].

[5] [Online]. Available: http://openrov.com/page/about. [Accessed 10 December 2012].

[6] D. Radu, M. French and B. Habin, "Design of Autonomous Underwater Vehicle and Optimization of Hydrodynamic Properties and Control," 28 4 2008. [Online]. Available: http://www.wpi.edu/Pubs/E-project/Available/E-project-042809-153324/unrestricted/Final_Report_sub@WPI09.pdf. [Accessed 22 4 2013].

[7] "Video Ray ROV," 2009. [Online]. Available: http://www.wesurveys.co.uk/ROV_1_25pc.jpg. [Accessed 4 April 2013].

[8] "Project: ORCA-2," 1999. [Online]. Available: http://web.mit.edu/orca/www/2000_comp1999.shtml. [Accessed 4 April 2013].

[9] "Autonomous operation of the Explosive Ordnance Disposal Robotic Work Package using CETUS untethered underwater vehicle," 10 Dec 2012. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=532396&tag=1. [Accessed 22 4 2013].

[10] National Commission on the BP Deepwater Horizon Spill and Offshore Drilling, "The History of Offshore Oil and Gas in the United States," [Online]. Available: https://docs.google.com/viewer?a=v&q=cache:L-XORMFenMoJ:www.eoearth.org/files/154601_154700/154673/historyofdrillingstaffpaper22.pdf+&hl=en&gl=us&pid=bl&srcid=ADGEESiw2DtN1iVzllRwUB1WIYQszAbgHRMClL8dxK3CUAVFPuZS8zmnf-52HocZKph53OQ_CqXQPdbTB_sVLt-T2fpCNcPB. [Accessed 11 December 2012].

[11] L. Allsop, "Hi-tech robots search ocean floor for ancient shipwrecks," [Online]. Available: http://articles.cnn.com/2010-11-18/tech/titanic.robots.shipwrecks_1_historic-shipwrecks-rms-titanic-ocean-floor?_s=PM:TECH. [Accessed 2012 Dec 11].

[12] "Robots dive into marine science," [Online]. Available: http://www.vims.edu/features/research/marine-robots.php. [Accessed 8 December 12].

[13] S. Gittings, "Remotely Operated Vehicles (ROVs)," [Online]. Available: http://oceanexplorer.noaa.gov/technology/subs/rov/rov.html. [Accessed 28 9 2012].

[14] "About us," [Online]. Available: http://www.auvsi.org/home/aboutus. [Accessed 29 September 2012].

[1 Association for Unmanned Vehicle Systems International, "History," [Online]. Available:
5] http://www.auvsi.org/Home/History. [Accessed 29 September 2012].

[1 AUVSI Foundation, "Robosub," [Online]. Available:
6] http://www.auvsifoundation.org/foundation/competitions/robosub. [Accessed 26 September 2012].

[1 V. P. Shah, "Design Considerations for Autonomous Underwater Vehicles," [Online]. Available:
7] http://dspace.mit.edu/bitstream/handle/1721.1/39893/182543456.pdf. [Accessed 29 September 2012].

[1 "Bluefin-21," 2013. [Online]. Available: http://www.bluefinrobotics.com/products/bluefin-21/.
8] [Accessed 17 4 2013].

[1 Woods Hole Oceanographic Institute, "SENTRY," [Online]. Available:
9] https://www.whoi.edu/fileserver.do?id=56044&pt=10&p=39047. [Accessed 20 4 2013].

[2 S. Dawicki, 2006 September 1. [Online]. Available: http://www.whoi.edu/main/news-
0] releases/2006?tid=3622&cid=16409. [Accessed 2013 20 4].

[2 A. Tarantola, "Bioswimmer," [Online]. Available: http://gizmodo.com/bioswimmer/. [Accessed 2013
1] April 20].

[2 C. Barngrover, "The Stingray Project," 29 4 2008. [Online]. Available:
2] http://cseweb.ucsd.edu/~cbarngrover/masters/research.html. [Accessed 22 4 2013].

[2 R. Boyle, "Coming Soon: Robot Sea Turtles That Carry Cargo in Their Shells," 10 4 2012. [Online].
3] Available: http://www.popsci.com/technology/article/2012-10/robot-sea-turtles-carrying-cargo-

their-shells-are-more-awesome-robot-fish. [Accessed 22 4 2013].

[2 FESTO, "AquaPenguin," [Online]. Available:
4] http://www.festo.com/rep/en_corp/assets/pdf/AquaPenguin_en.pdf. [Accessed 22 4 2013].

[2 AUVSI and ONR, "Engineering Primer for AUV Team Competition," July 2007. [Online]. Available:
5] http://higherlocifdownload.s3.amazonaws.com/AUVSI/fb9a8da0-2ac8-42d1-a11e-d58c1e158347/UploadedImages/Support_Primer_r1.pdf. [Accessed 29 September 2012].

[2 "DRDO," 30 January 2012. [Online]. Available: http://spsmai.com/exclusive/?id=31&q=DRDO-ready-
6] to-demonstrate-indigenous-AUV. [Accessed 4 April 2013].

[2 L. Billings, "Deep-sea Discoveries on Expedition Using ASTEP AUVs," 8 July 2008. [Online]. Available:
7] http://astrobiology2.arc.nasa.gov/articles/deep-sea-discoveries-on-expedition-using-astep-auvs/. [Accessed 14 March 2013].

[2 "MOHAWK," [Online]. Available: http://www.f-e-t.com/our_products_technologies/subsea-
8] solutions/rovs-observation/mohawk/. [Accessed 4 April 2013].

[2 AUVSI/ONR, "AUVSI/ONR Engineering Primer Document," AUVSI/ONR, 2007.
9]

[3 IRobot, "1KA Seaglider," [Online]. Available:
0] http://www.irobot.com/us/robots/Maritime/Seaglider.aspx. [Accessed 29 September 2012].

[3 Wm. Olds & Sons Pty. Ltd., "ROV and AUV Thrusters," [Online]. Available:
1] http://www.olds.com.au/marine/ROV_AUV_Thrusters.html. [Accessed 22 April 2013].

[3 U. o. Washington, "Applied Physics Laboratory, University of Washington," [Online]. Available:
2] http://www.apl.washington.edu/projects/seaglider/summary.html. [Accessed 22 April 2013].

[3 Applied Physics Laboratory, "Seaglider," [Online]. Available:
3] http://www.apl.washington.edu/projects/seaglider/summary.html. [Accessed 22 April 2013].

[3 A. Tarantola, "Is This Tuna-Bot the Future of US Harbor Security?," 21 September 2012. [Online].
4] Available: http://gizmodo.com/5945095/is-this-tuna+bot-the-future-of-us-harbor-security. [Accessed 22 April 2013].

[3 Alcen, "SeaExplorer," [Online]. Available: http://acsa-alcen.com/robotics/seaexplorer. [Accessed 28
5] November 2012].

[3 B. Martin, "Model Submarine Diving Technology," [Online]. Available: http://www.rc-
6] sub.com/resources/index.php5. [Accessed 22 April 2013].

[3 P. Chotikarn, W. Koedsin, B. Phongdara and P. Aiyarak, "Low Cost Submarine Robot," Songklanakarin
7] Journal of Science and Technology, Songkla, 2010.

[3 Bluefin Robotics, "Energy," [Online]. Available: http://www.bluefinrobotics.com/technology/energy.
8] [Accessed 29 September 2012].

[3 X. Wang, "Review of Power Systems and Environmental Energy Conversion for unmanned underwater
9] vehicles," [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032111006095
. [Accessed 18 September 2012].

[4 "TAUVROS SAUV Configuration," [Online]. Available: http://auvac.org/configurations/view/222.

0] [Accessed 28 November 2012].

[4 "First Underwater AUV Powered Entirely by Ocean's Thermal Energy," [Online]. Available:
1] http://www.rovworld.com/article4168.html. [Accessed 29 November 2012].

[4 B. University, "Comparison Tables of Secondary Batteries," Battery University, [Online]. Available:
2] http://batteryuniversity.com/learn/article/secondary_batteries. [Accessed 20 September 2012].

[4 L. A. Gish, "Design of an AUV Recharging System," [Online]. Available:
3] http://dspace.mit.edu/bitstream/handle/1721.1/33445/62887131.pdf. [Accessed 18 April 2013].

[4 Microsoft, "Product Information," [Online]. Available: http://www.microsoft.com/robotics/#Product.
4] [Accessed 22 April 2013].

[4 M. Somby, "A Review of Robotics Software Platforms," [Online]. Available:
5] http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/A-review-of-robotics-software-platforms. [Accessed 28 September 2012].

[4 "Robotic Sub running Devian wins International Competition," [Online]. Available:
6] http://cuauv.ece.cornell.edu/node/1848/. [Accessed 2 October 2012].

[4 ROS.org, "About ros.org," [Online]. Available: http://www.ros.org/wiki/About%20ros.org. [Accessed
7] 22 April 2013].

[4 Neuron Robotics, "Bowler Communications System," 16 May 2012. [Online]. Available:
8] http://wiki.neuronrobotics.com/Bowler_Communications_System. [Accessed 22 April 2013].

[4 E. B. A. Y. N. Morgan Quigley, "STAIR: Hardware and Software Architecture," *AAAI,* 2007.

9]

[5 S. U. Computer Science Department, "ROS: An Open-Source Robot Operating System.," September 28
0] 2012. [Online]. Available: http://pub1.willogarage.com/~konolige/cs225B/docs/Quigley-icra2009-
ros.pdf.

[5 W. Contributors, "XML-RPC," 27 September 2012. [Online]. Available:
1] <http://en.wikipedia.org/wiki/XML-RPC>.

[5 Neuron Robotics, "About Us," [Online]. Available: http://www.neuronrobotics.com/about/.
2]

[5 J. R. Welty, C. E. Wicks, R. E. Wilson and G. L. Rorrer, Fundamentals of Momentum, Heat Transfer, and
3] Mass Transfer, 5th edition, Jon Wiley & Sons, Inc., 2007.

[5 "ABout Trolling Motors," [Online]. Available: www.trolling-motor.info . [Accessed 2013 4 22].
4]

[5 S. Fitzgerald, "Keyboard Logout," Arduino, 7 April 2012. [Online]. Available:
5] http://arduino.cc/en/Tutorial/KeyboardLogout. [Accessed 22 April 2013].

[5 "Neuron Robotics Development Kit," 2012. [Online]. Available:
6] http://www.neuronrobotics.com/neuronrobotics/store/nrdk/.

[5 B. Systems, "IP (Ingress Protection) Ratings," 2013. [Online]. Available:
7] http://www.bluesea.com/viewresource/117 . [Accessed 10 April 2013].

[5 SINTEF, "Robots Taking Over The Job On Offshore Oil Drilling Platforms," 1 January 2008. [Online].

8] Available: http://www.sciencedaily.com/releases/2007/12/071221230852.htm. [Accessed 10 December 2012].

[5 John Hopkins University, "Department of Engineering," [Online]. Available:
9] http://www.me.jhu.edu/r_ocean.html. [Accessed 10 December 2012].

Appendix A: Chassis Design Matrix

The weights for each item are determined in Table 19. Each individual rated each criterion. The average of these values was used as the weight for that item.

Table 19: Chassis Design Matrix - Weight Determination

Mechanical					Electrical				Software			Avg	Criteria
AC	CL	CO	LM	SB	AV	BM	IE	NS	AT	DM	EO		
8	9		8	6				9		7	7	8	Battery Access
10	9		8	5				10		6	9	8	Electronics Housing Access
8	8		8	9				9		8	6	8	Module Placement
8	8		9	8				8		6	8	8	Part Replacement
5	4		7	5				7		7	5	6	Waterside approach/retrieval
7	6		9	4				8		6	7	7	Ease of manufacturing
4	4		3	3				7		7	3	4	Hydrodynamic shrouding
3	3		5	3				1		5	3	3	Hydrodynamics without many modules
	3		3	6				10		5	1	5	Hydrodynamics with many modules
9	8		5	4				5		6	7	6	Level of design simplicity
8	6		7	7				8		8	5	7	Ruggedness
5	6		7	7				7		8	7	7	Buoyancy Redistribution
6	3		4	6				1		2	4	4	Rule of Cool
10	7		4	2				9		4	6	6	Gut Feeling

Each team rated the three designs based on the criteria. Table 20 shows Boxy, Table 21 shows Roddy and Table 22 shows Octopuck.

Table 20: Chassis Design Matrix - Boxy

Mechanical					Electrical				Software			Avg	W. Avg	Criteria
AC	CL	CO	LM	SB	AV	BM	IE	NS	AT	DM	EO			
8			8	7	10		8	9	8	8	4	7.8	60.0	Battery Access
10			8	5	10		6	9	9	7	6	7.8	63.3	Electronics Housing Access
7			8	7	10		7	8	6	8	7	7.6	60.4	Module Placement
7			8	10	10		5	8	7	7	5	7.4	58.5	Part Replacement
10			8	6	10		8	9	7	7	5	7.8	44.4	Waterside approach/retrieval
10			9	10	10		8	10	8	8	9	9.1	61.2	Ease of manufacturing
5			4	7	9		6	7	9	8	8	7.0	31.0	Hydrodynamic shrouding
5			5	5	8		7	7	6	7	5	6.1	20.1	Hydrodynamics without many modules
			3	3			7	5	6	7	4	5.0	23.3	Hydrodynamics with many modules
7			8	7	10		8	8	8	8	9	8.1	51.0	Level of design simplicity
10			7	7	10		7	9	8	9	8	8.3	58.3	Ruggedness
9			7	8	10		8	9	9	8	8	8.4	56.7	Buoyancy Redistribution
4			2	2	10		4	7	5	8	4	5.1	19.0	Rule of Cool
7			8	7	10		5	8	6	10	6	7.4	44.7	Gut Feeling
												103	651.97	

Table 21: Chassis Design Matrix - Roddy

ME						ECE					CS		Avg	W. Avg	Criteria
AC	CL	CO	LM	SB	AV	BM	IE	NS	AT	DM	EO				
6			8	7	5		7	5	6	6	7	6.3	48.9	Battery Access
6			8	5	6		6	7	4	6	8	6.2	50.7	Electronics Housing Access
6			7	5	5		6	5	5	5	8	5.8	46.2	Module Placement
5			5	3	5		7	5	5	5	8	5.3	41.9	Part Replacement
10			8	7	7		7	8	6	4	4	6.8	38.7	Waterside approach/retrieval
9			8	4	9		8	8	7	7	5	7.2	48.5	Ease of manufacturing
10			10	3	7		7	9	8	6	3	7.0	31.0	Hydrodynamic shrouding
10			9	7	7		6	10	8	8	4	7.7	25.2	Hydrodynamics without many modules
10			8	4	7		6	8	8	8	2	6.8	31.6	Hydrodynamics with many modules
10			8	5	7		8	9	7	6	4	7.1	44.7	Level of design simplicity
7			6	4	7		3	6	5	4	3	5.0	35.0	Ruggedness
8			10	2	7		6	8	8	7	4	6.7	44.8	Buoyancy Redistribution
5			9	7	5		1	7	5	6	8	5.9	21.9	Rule of Cool
6			8	5	5		1	5	7	6	3	5.1	30.7	Gut Feeling
												89	539.69	

Table 22: Chassis Design Matrix - Octopuck

ME						ECE				CS		Avg	W. Avg	Criteria
AC	CL	CO	LM	SB	AV	BM	IE	NS	AT	DM	EO			
8			8	6	9		2	9	8	7	4	6.8	42.9	Battery Access
8			8	5	7		2	9	8	6	4	6.3	39.4	Electronics Housing Access
8			8	9	7		7	8	8	7	7	7.7	44.3	Module Placement
8			7	8	6		7	8	8	7	5	7.1	37.9	Part Replacement
8			7	5	8		7	9	7	7	5	7.0	47.4	Waterside approach/retrieval
7			6	6	8		6	7	7	6	8	6.8	49.0	Ease of manufacturing
7			5	6	6		7	8	8	5	7	6.6	45.9	Hydrodynamic shrouding
6			5	7	6		5	8	8	5	6	6.2	47.7	Hydrodynamics without many modules
6			4	6	6		5	7	6	5	5	5.6	37.7	Hydrodynamics with many modules
6			5	5	7		6	8	8	5	7	6.3	45.0	Level of design simplicity
8			7	6	8		4	9	8	7	8	7.2	36.1	Ruggedness
9			9	10	8		7	10	9	7	7	8.4	56.3	Buoyancy Redistribution
8			5	8	4		1	10	5	4	6	5.7	33.4	Rule of Cool
8			7	6	4		2	9	6	4	7	5.9	30.1	Gut Feeling
												94	593.11	

Appendix B: Ballast Design Matrix

This design matrix uses a scale of 1 to 5, where 1 is bad and 5 is good. Cost is the estimated expense to create the system. The manufacturability is how easy it is to fabricate the system. The in-mission flexibility is the ability of the system to trim while in the water. Payload range is how much control of the buoyancy system there is before the craft is placed in the water.

Table 23: Ballast Design Matrix

Weights		Anna	Chris	Cory	Lisa	Sidney	Total	Weighted total
	Full Piston Design							
2	Cost	2	2	4	2	2	12	24
3	Manufacturability	2	2	2	2	2	10	30
1	In mission flexibility (trimming)	4	3	4	3	5	19	19
4	Payload Range	2	3	2	2	3	12	48
								Total: 121
	Full Foam Design							
2	Cost	5	4	3	3	4	19	38
3	Manufacturability	5	4	4	3	5	21	63
1	In mission flexibility (trimming)	1	1	1	1	1	7	7
4	Payload Range	5	4	4	3	4	20	80
								TOTAL: 186
	Hybrid Design							
2	Cost	3	4	5	4	3	19	38
3	Manufacturability	3	4	4	3	3	17	51
1	In mission flexibility (trimming)	3	4	5	5	3	20	20
4	Payload Range	3	5	4	5	3	20	80
								TOTAL: 189

Appendix C: fit-PC Feature Comparison

Table 24: Fit-PC Comparison

Device	CPU	RAM	HDD	OS	Power Connection	Watts	Base Price	Extras	Total
Fit-PC2i Linux	1.6 GHz	2 GB	250 GB	Linux Mint	12 Volt	6W low/8W full/1W standby	397	0	397
Fit-PC2i Diskless	1.6 GHz	2 GB	None	None	12 Volt	6W low/8W full/1W standby	325	50	375
Fit-PC2i SSD	1.6 GHz	1 GB	16GB SSD	None	12 Volt	6W low/8W full/1W standby	352	0	352
Fit-PC3 LP Barebone	1 GHz 2core	None	None	None	Unregulated 10-16Volt	7-15W	381	80	461
Fit-PC3 Basic 4GB	1 GHz 2 core	4 GB	None	None	Unregulated 10-16Volt	8-17W	403	50	453
Fit-PC3 Basic Barebone	1 GHz	None	None	None	Unregulated 10-16Volt	8-17W	369	80	449
Fit-PC3 value Barebone	1.2 GHz	None	None	None	Unregulated 10-16Volt	8-17W	328	80	408

The extras considered were 2 GB of RAM for $30, and a 32 GB solid state drive for $50. The Fit-PC3 Basic 4GB was chosen, and a 64 GB solid state drive was purchased for only slightly more than the 32GB drive would have been. This PC was chosen due to it being the most powerful model that also fit WAVE's power constraints.

Appendix D: Waterside Deployment and Recovery SOP

D1. Preparations

1. Ensure that the deployment team is allowed to use intended body of water ahead of time
2. State launch intentions and day of deployment to whoever may be in charge of the body of water
3. If anyone will be in the water with the UUV, know ahead of time who they will be
4. Make sure that any batteries are charged
5. Know what configuration is intended to be run and plan the UUV's passive ballast accordingly.
    a. If it will be the UUV's first water test using a new module configuration, see step 13 below.

D2. Deployment

1. Transport the UUV to the launch site
2. Inspect the UUV for damage or anomalies
    a. Are the end-cap seals clean? If not, clean them. Remove debris such as hair or sand. Compressed air and/or alcohol swabs can help here
    b. Are components securely fastened? Try jiggling them. Check screws and other metal components for harmful corrosion. If so, …???
    c. Do the LiPo batteries look "puffy"? If so, that's very bad. Do not use. For more info on battery safety, reference Appendix E1.1 Charging Safety IMPORTANT!.
    d. Before inserting the electronics rack into the housing, visually inspect the boards and wires. Are there any burnt spots or areas that look like they may short? Use the GUI and warning LEDs to determine if there is a problem. If so, carefully inspect each component and identify the problem(s). Regardless, keep the electronics rack outside of the housing.
    e. Is there any sign of moisture inside the electronics housing? If so, carefully dry the interior. Leave end-caps off until completely dry.
3. Install the batteries
4. Power up the computer and electronics

5. Perform a computer systems check
    a. Is the computer responding? Check if the fit-PC's lights are on and if the CPU is communicating with the poolside user interface.
    b. Is the correct subsystems configuration loaded? Check the GUI to make sure.
    c. Are online systems nominal? As with the previous checks, use the GUI to ensure no problems are occurring.
6. Connect whatever wires remain to be connected between the end-cap and the electronics rack
7. Enable power and slide the Electronics Rack into its housing
8. Double check that the end-cap seals are clean
9. Seal the main electronics housing. Make sure the end-caps are tightly fastened using the clasps.
10. Perform a second check system check to ensure that high power subsystems report nominal
    a. Pulse very briefly the thrusters to ensure that they do indeed work.
11. Notify the nearby vicinity that the UUV is being put into the water
    a. If it is being carried or hauled into the water by hand, have two people handle the sub and a third person spotting or overseeing if possible.
    b. If a crane or skyhook is being used, secure the four top corners of the sub using the provided four-strap system. Have one person steady the UUV as it is lifted and moved. Keep the crane operator in communication with the steadier. Once in-water, let the lift-straps hang loose.
12. Make sure the UUV levels itself out. Active ballast should try to do this on its own
    a. If it is the first or second time the particular module configuration is used, the UUV should be deployed in calm shallow water, ~1.2m (4 ft) deep, and two people should be in the water with it to adjust passive ballast. The craft should be approximately neutrally buoyant, approximately level in pitch (tilt up/down), and as level as possible in roll (tilt side to side).
    b. If a crane was used, remove the support straps.
13. WAVE is now mission ready

D3. Recovery

1. Remove the craft from the water and place it in its transport cradle.

a. Warning - the housing may be hot.

b. One or two people will have to be in or near the water to either retrieve the craft or hook it up to the crane.

2. Thoroughly rinse the UUV off with fresh/ hose water
3. Retreat from the immediate waterside location
4. Dry off the UUV somewhat (stop drippings) with particular attention to the end-cap areas and wires
5. Open the electronics housing, taking care not to drip onto its contents.
6. Switch off the high current switch
7. Download log files and any other data from the computer then power it all down such that the batteries may be safely removed
8. Remove the batteries. Set them aside for cool down or charging (in a dry area!)
9. Secure the UUV and its components for transport
10. Notify body of water's authority that you have concluded your session

Appendix E: User Manuals

E1. How to Safely Use Lithium Polymer Batteries

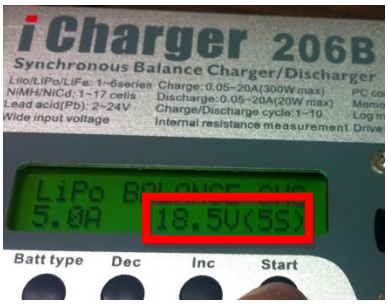
E1.1 Charging Safety IMPORTANT!

The primary cause of lithium polymer fires is due to overcharging, so follow these instructions carefully!

1. **Use charger that is approved for lithium batteries.** The charger may be designed for Li-Ion or LiPo, because they charge in the same way. Do NOT use NiCd or NiMH chargers. Charging is one of the most hazardous parts of using lithium batteries; this is why using the correct charger is the first step to safety.
2. **Inspect batteries visually before each charge.** Ensure there are no physical defects in the balancing connections, wires, or packaging.
3. **Place batteries into LiPo safety bag.** In the case that an explosion does happen, the fire will be (mostly if not completely) contained.

4. **Use a safe surface to charge your batteries.** If explosion does occur, the least amount of damage will be done.
5. **NEVER charge the batteries unattended.** Always monitor the batteries while charging. If batteries start to puff or if smoke is seen, promptly disconnect the batteries. If a cell balloons quickly, place it in a fire safe place if possible. After you have let the cell sit in the fire safe place for at least 2 hours. Discharge the cell/pack slowly to 3V per cell and then throw the battery away.

6. **Ensure that the cell count on the charger is correct.** The 10000mAh Gens Ace LiPos are 5S, meaning they are made of 5 individual cells connected in series within the pack. Ensure that the charger reflects 5 cells.



7. **Watch the charger very closely for the first few minutes of operation to ensure that the correct cell count continues to be displayed.** If the cell count changes, stop charging, reset the charger information, and try again while continuing to closely monitor the charger and battery.

8. **Check the voltage of each individual cell on the charger and ensure they are equal.** If they are not within 0.2V of each other, then they are not equal and should be balanced before proceeding to charge. If they are not properly balanced before charging, over discharging may happen and the battery may explode even if correct cell count is chosen on charger. Do not measure individual voltages from balancing plug using volt meter probes because cells might accidentally be shorted in the process. Note: If the pack is unbalanced after every discharge, a cell is faulty and the battery pack should be replaced.

9. **Charge at 1-3C as recommended from the battery manufacturer.** The batteries are 10000mAh; therefore, 1C=10A, 2C=20A, and 3C=30A. Do not exceed 30A.

10. **Do NOT exceed 4.2V for each individual cell and do not drain any cell under 3.0V (without load).** These are the maximum and minimum operating conditions for each LiPo cell. Over charging may lead to thermal runaway and cell rupture, which can lead to combustion. Over discharging may short a cell, which can also lead to combustion.

11. **Do NOT fully charge in cold temperatures.** Voltage increases with temperature. If the batteries are fully charged in a colder temperature and then moved to a warmer temperature, this will have the same effect as over discharging the batteries and can result in battery damage and explosions.

12. **DO NOT puncture the cell, ever.** It may puff quickly and then explode. If a cell balloons quickly, place it in a fire safe place <u>if possible</u>. After you have let the cell sit in the fire safe place for at least 2 hours. Discharge the cell/pack slowly to 3V per cell and then throw the battery away.

13. **Keep batteries from being struck in any way.** If batteries are jostled, through a crash, through being dropped, etc., carefully monitor the batteries' behavior for at least 20 minutes. The

batteries may appear to have no damage, but it is possible that they are shorted inside. Ensure that they are behaving normally.

14. **Avoid water.** Though the packs are water proof, avoid getting water on them. Pure water is non-conductive and will not do anything to the batteries, but pool water can be conductive and if both terminals are submerged, the battery will short and possibly lead to combustion. If batteries do get wet with, dry them completely before use.

15. **Charge your batteries in an open ventilated area.** If a battery does rupture or explode, hazardous fumes will spew from the battery.

16. **Keep an ABC fire extinguisher nearby, but not too close to the batteries.** In the event that a battery does explode and fire escapes the LiPo safety bag, use the fire extinguisher. The LiPo battery itself will continue to burn until the battery has no energy left, regardless of the extinguisher; however, the extinguisher will keep the fire from spreading. Keep the extinguisher close enough that it can quickly be obtained to respond to the fire, but not so close that it is within the flames of the fire itself!

17. **Do not charge near flammable substances.**

18. **Do not think "it won't happen to me". ** This thought leads to not following the proper procedure for safely using LiPo and is the underlying cause for many LiPo fires.

E1.2 Guidelines for Storage and Transportation

1. **Fully charge batteries, and then discharge between 50%-60% of their capacity for long term storage.** Discharge to 30% for flights.

2. **Keep at room temperature ideally between 0°C - 21°C, 40°F- 70°F.** Do not exceed 120°F.

3. **Do not leave in direct sunlight for extended periods of time.**

4. **Always store in LiPo safety bag or fire proof container.** Never leave them loosely lying around.

5. **Ensure all connectors are covered.** This prevents accidental shorts which could lead to combustion.

6. **Do not store next to combustible material.**

7. **Never leave in vehicle indefinitely.** Temperature of vehicle can easily rise above LiPo safety temperature of 120°F.

E1.3 Guidelines for Battery Disposal

1. **Discharge to 0V with a low load across the pack.**
2. **Slice small incision through the outer pouch along the edge where the blade cannot accidentally go into the plates.**
3. **Drop the pack with incision into a container of salt water to completely discharge the pack.**
4. **Throw battery away.**

E1.4 Detailed Steps for Charging

Turn Power Supply on.

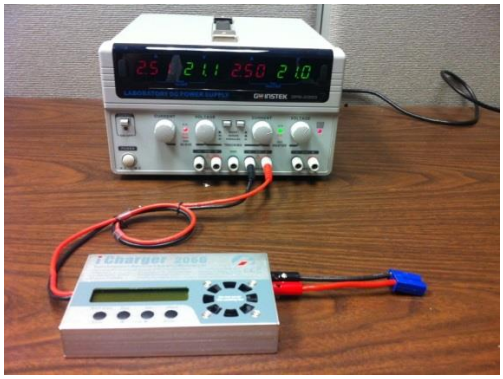
Press the two middle buttons in to configure the channels in parallel.

Use right current and voltage knobs to set the output to 21V and 2.5A (for a total of 5A).

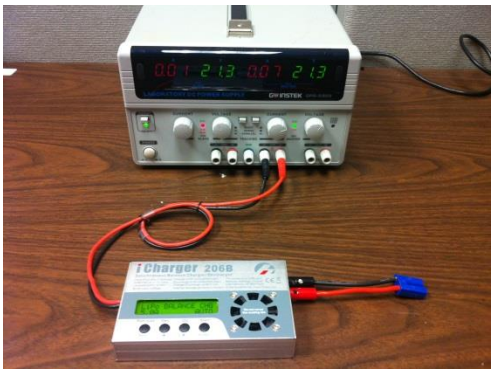
Get the charger.

Plug in the battery connection power cables.

Plug the power cables of the charger into the power supply.

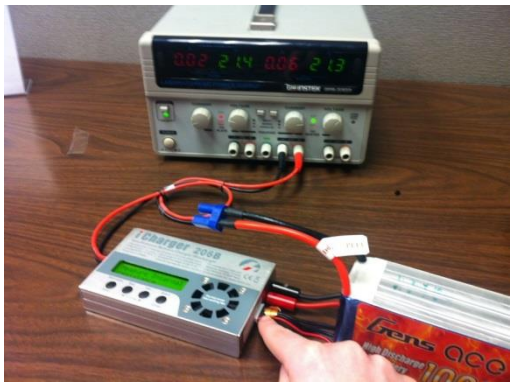


Turn Power Output on. iCharger will turn on. The screen will light up with a display.



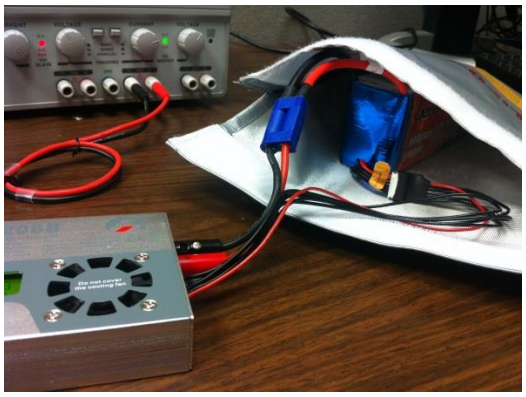
Connect the battery extension to the battery balancing plug.

Insert the extended balancing plug into the charger.



Connect the battery to the charger.

Place battery into the LiPo safety bag.

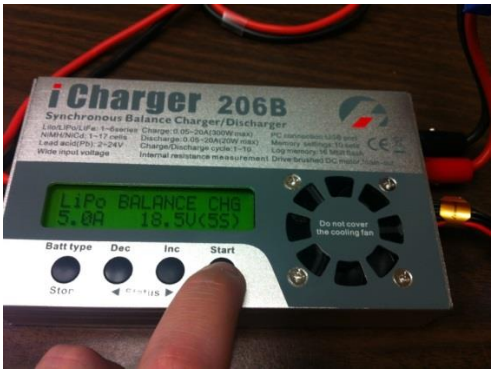


Tuck wires into the side of the pouch closing.

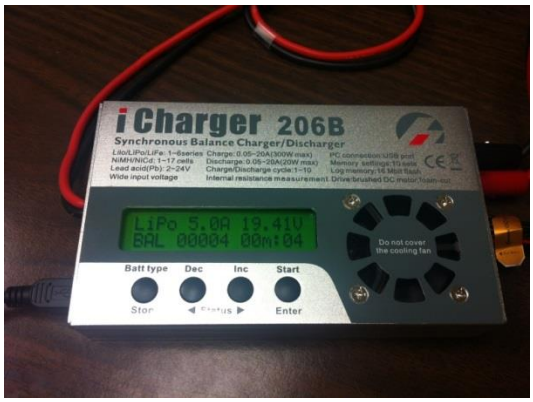
Seal the bag closed.

Press “Inc” until the “PROGRAM SELECT Lithium battery” screen is displayed.

Press “Inc” until “LiPo BALANCE CHG” screen appears. Ensure that the screen displays the 18.5V(5S) option for charging WAVE’s batteries. Also, ensure everything is connected properly and voltage and current is correct. Hold “Start” for 3 seconds to begin charging.



Screen will look like below.



Press “Inc” to view voltage of individual cells.

Press “Stop” when LiPo reaches 21V (4.2V per cell). After pressing “Stop”, disconnect the battery from the charger battery connection—do NOT pull plugs from charger! Then disconnect the balancer, and then turn the power off to the iCharger. Put everything back in iCharger box when finished.

Every 10 Cycles Check the Internal Resistance to monitor aging effects on battery.

Hit leftmost button until “Special modes” is displayed.

Press the Start/Enter button. The MOTOR DRV screen will display. Press “Inc” until the “Measure Internal resistance” screen is displayed. Hold “Start” down for 3 seconds.

The screen below appears, showing the internal resistance of each cell.

E2. LPCXpresso Manuals

E2.1 Installation and Registration

1. Before beginning the installation process, please check to make sure that your device meets the following requirements:

    - Operating System
        - Microsoft Windows –
            - XP 32-bit (SP2 or greater)
            - Vista 32-bit or 64-bit
            - Windows 7 32-bit or 64-bit
            - Windows 8 32-bit or 64-bit
        - Mac OS X
            - 10.7 (Lion)
            - 10.8 (Mountain Lion)
        - Linux
            - Ubuntu 9
            - Ubuntu 10
            - Ubuntu 11
            - Fedora 12

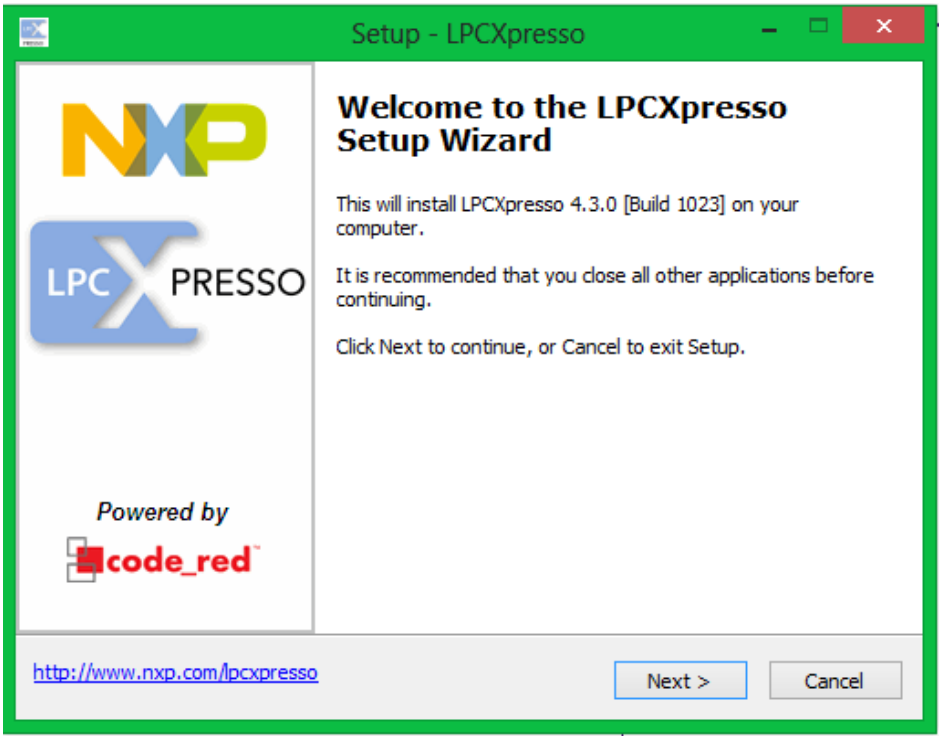
        - Fedora 13

- System RAM
    - 512 MB minimum
    - 1 GB recommended
- Hard Disk
    - 300+ MB of available space
- Screen/Display Adaptor
    - 1024 by 768 minimum recommended
- Internet Connection
    - High-speed internet to download and register software

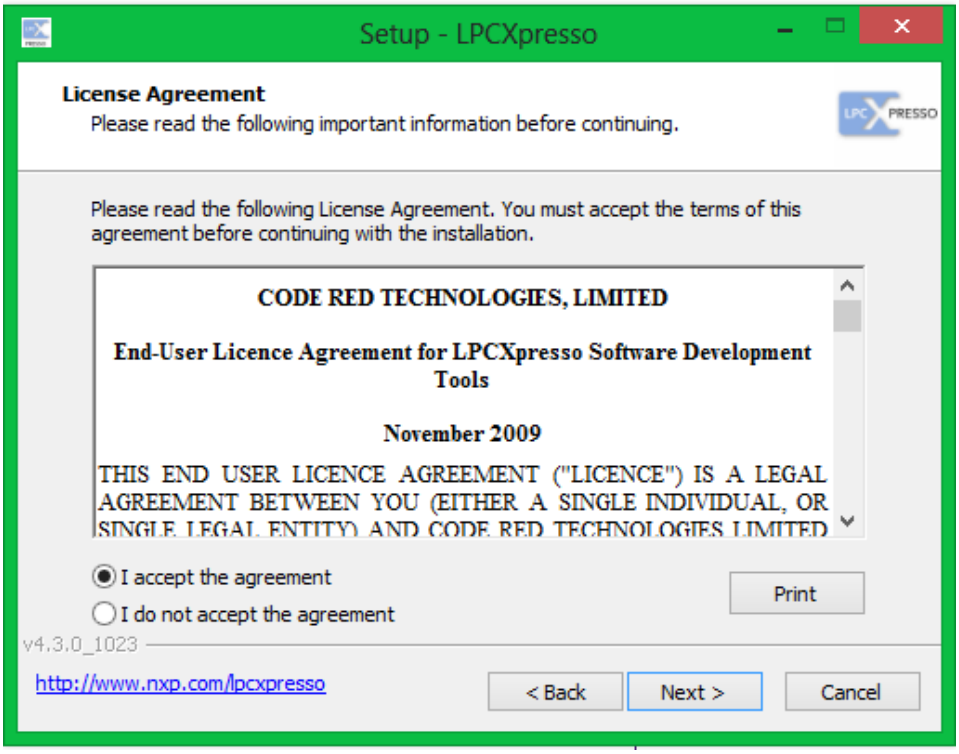
2. Once you have confirmed that your device is compatible, go to the Code Red website and create an LPCXpresso account, which is necessary to download and register a copy of LPCXpresso.

   http://lpcxpresso.code-red-tech.com/LPCXpresso/

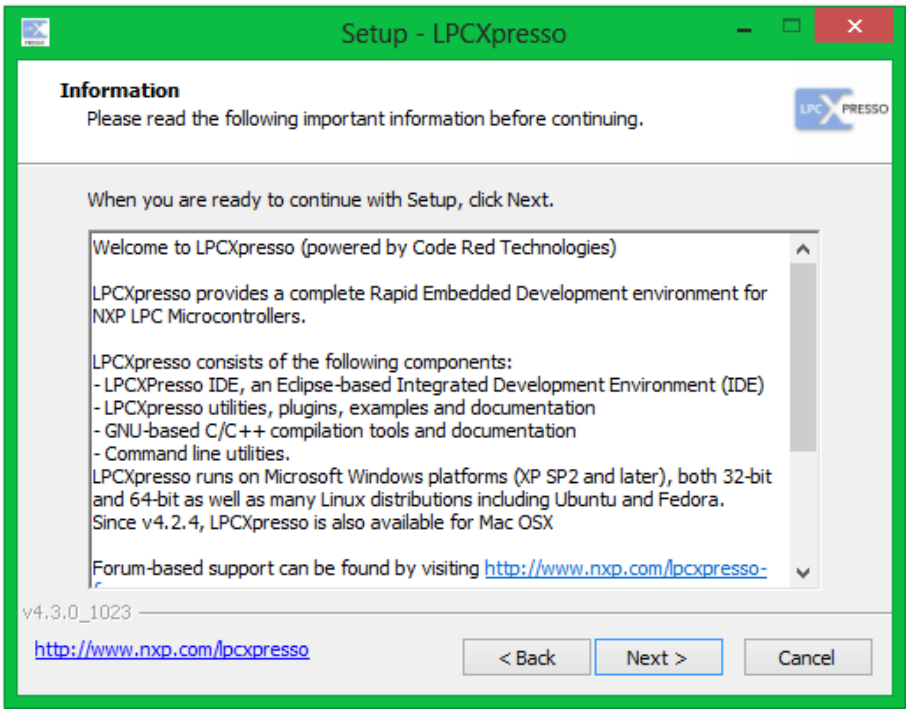
3. Now that you have your own LPCXpresso account, you will able to download the installer file. Under downloads choose LPCXpresso 4 specific to your operating system. (Note: to install for Linux, the download file needs to be marked as executable using chmod +r). The installation wizard will open. For the first prompt, simply choose **Next**.



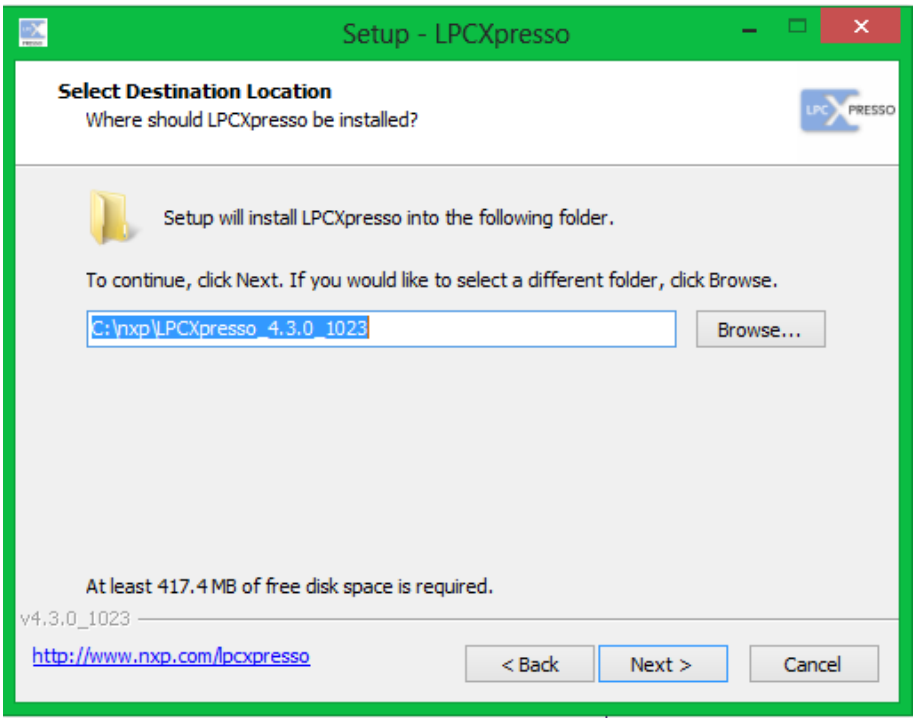
4. Next you will need to read and accept the License Agreement. Then click **Next**.



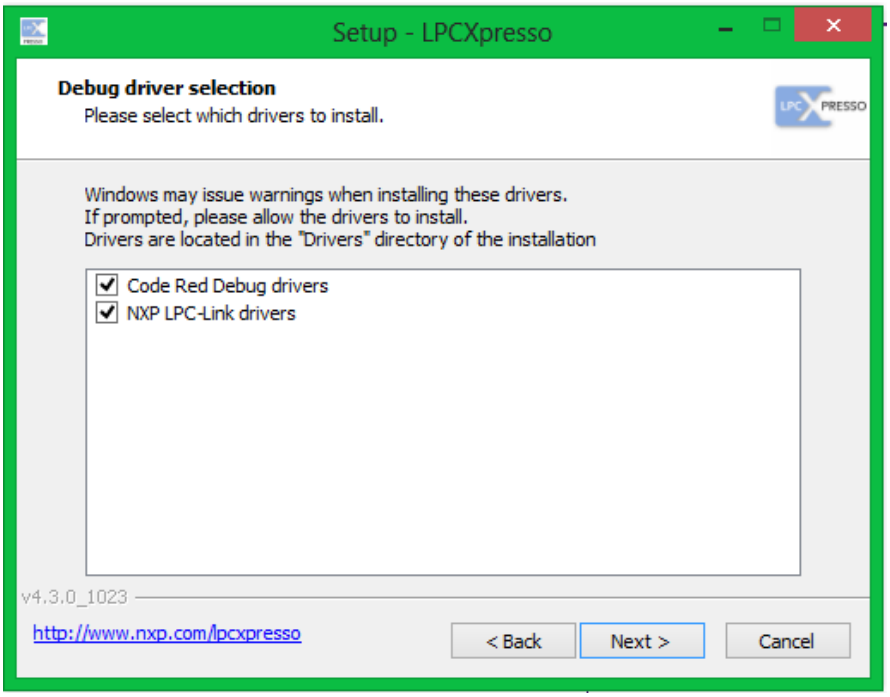
5. You will be provided with information regarding LPCXpresso. Read the information then click **Next**.



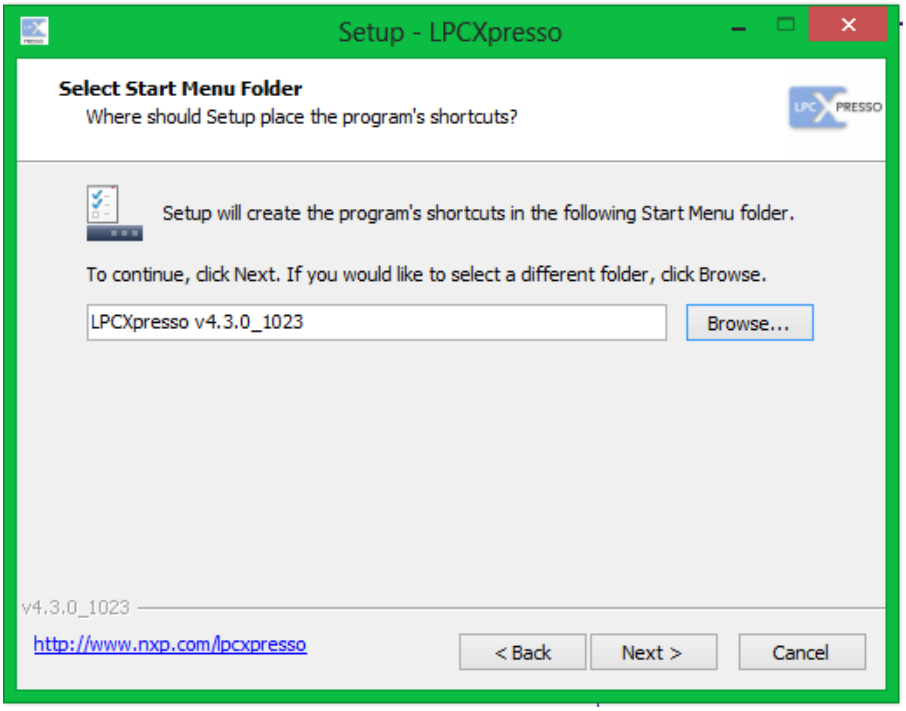
6. Now you must choose a destination location for LPCXpresso. You can use the default location or select **Browse** to choose another location. When a destination has been chosen, click **Next**.



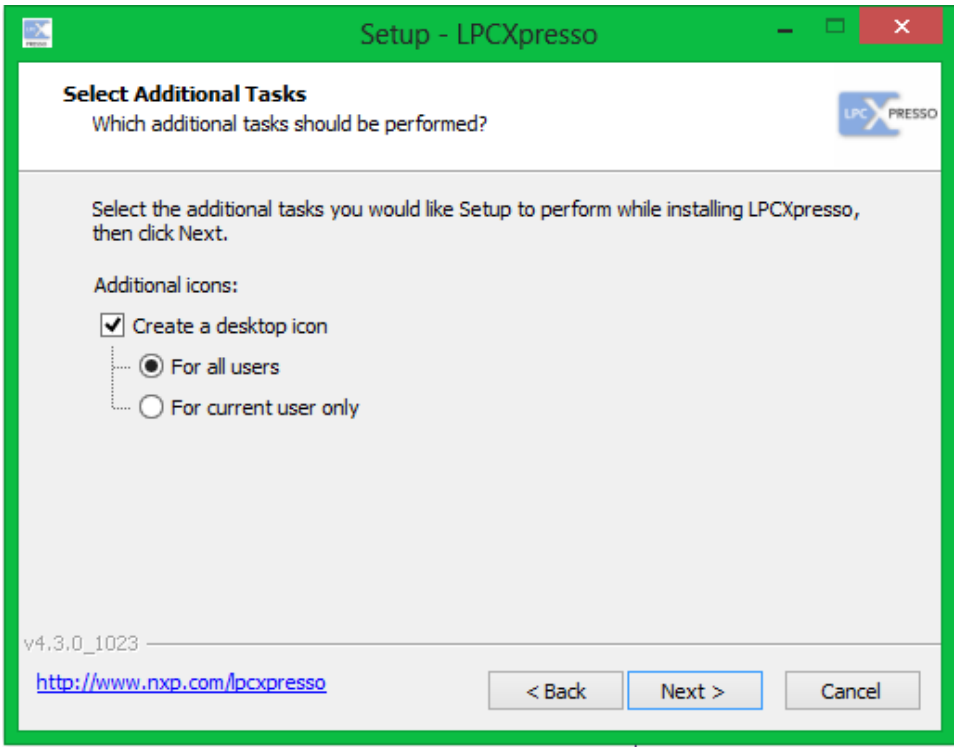
7. Now you must choose which debug drivers to install. Select *Code Red Debug drivers* and *NXP LPC-Link drivers* by checking the boxes to the left of them. Then click **Next**.



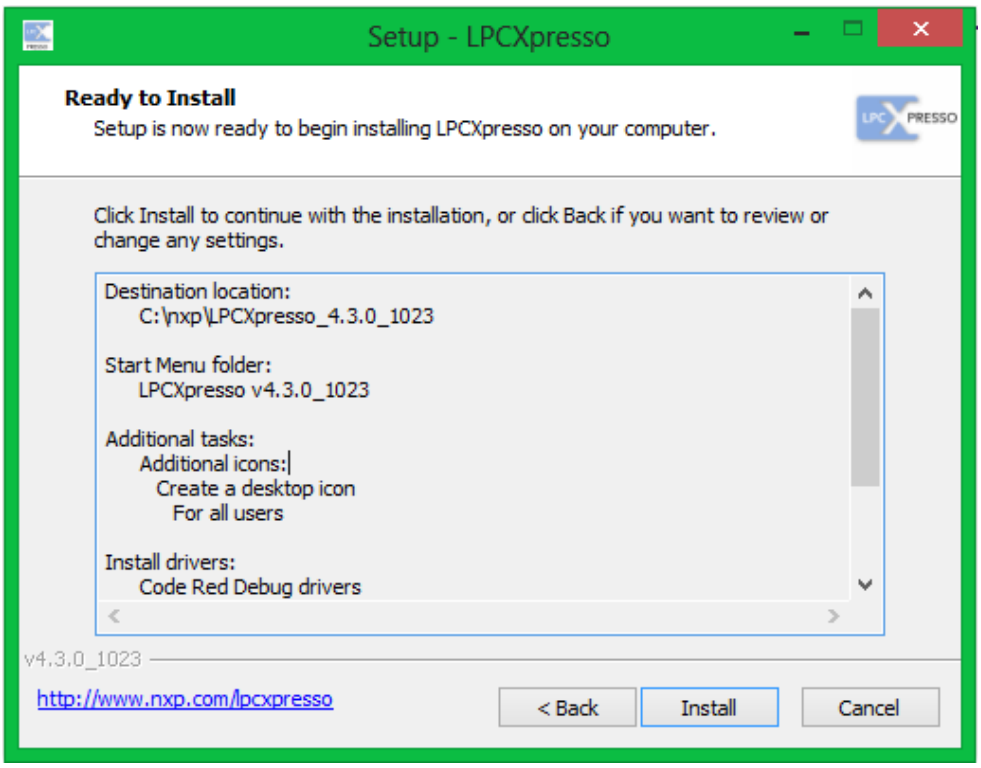
8. You will then be prompted to select the start menu folder. It is recommended that you use the default location, which is the destination folder from earlier, but if you want to choose a different location choose **Browse**. Afterwards click **Next**.



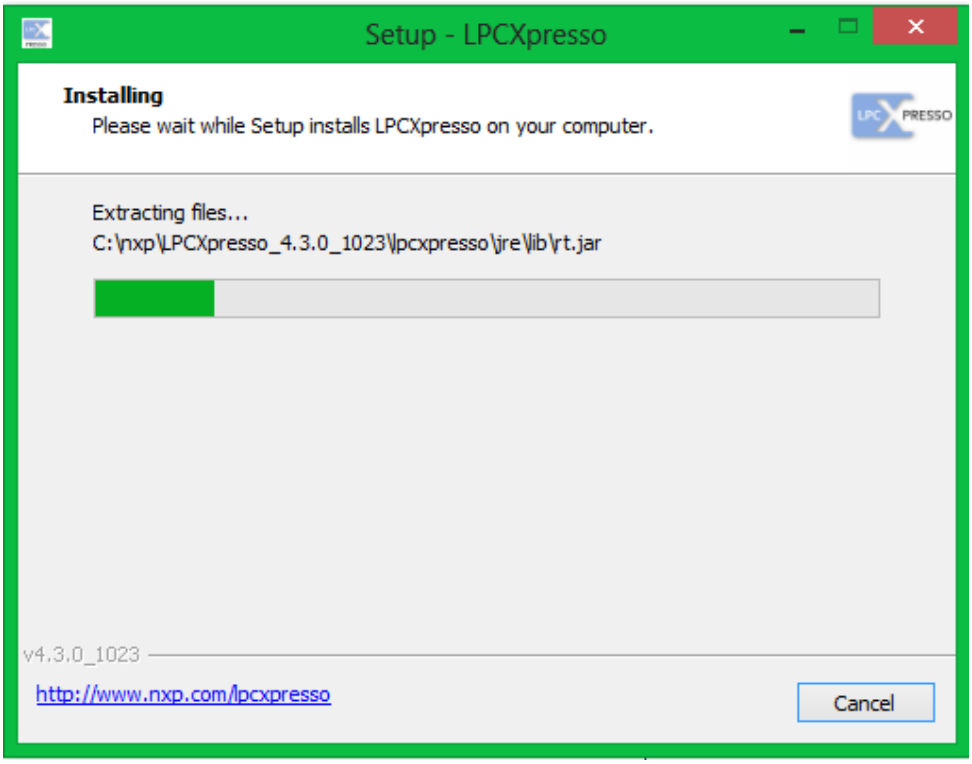
9. You will be provided with the opportunity to select additional tasks for the installation process. Choose whichever tasks you desire and click **Next**.



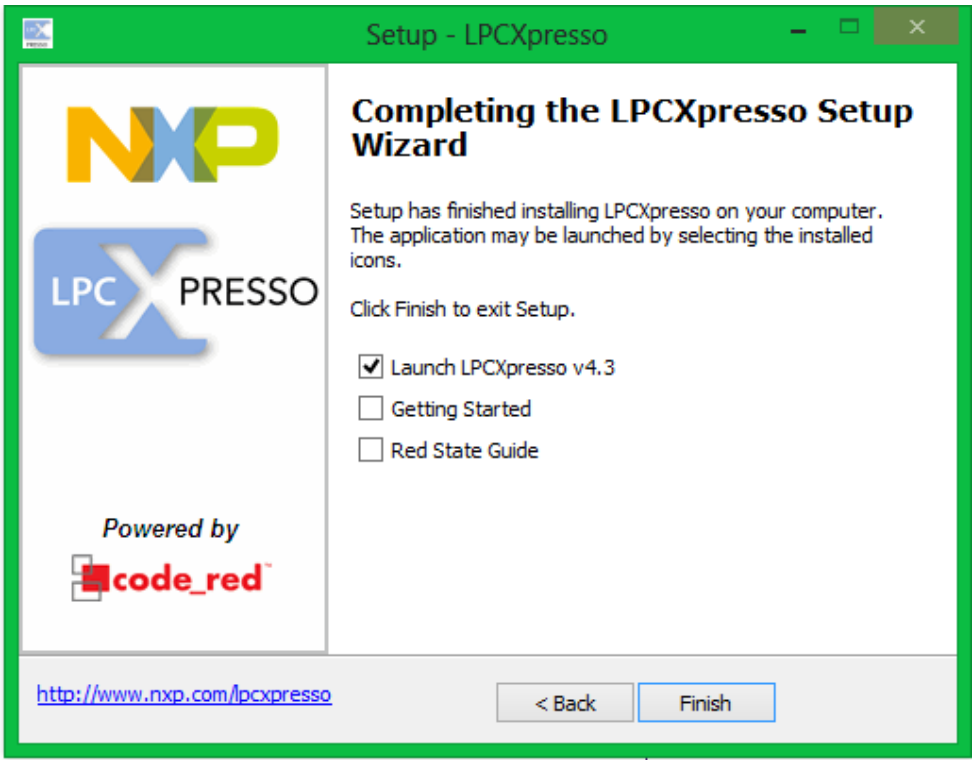
10. Review the summary information on the next pane. When you are ready, click **Install**.



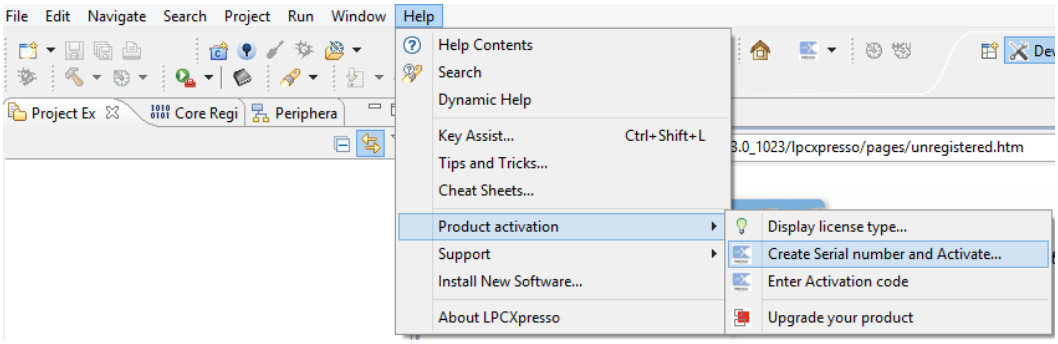
11. Wait patiently while LPCXpresso is installed.



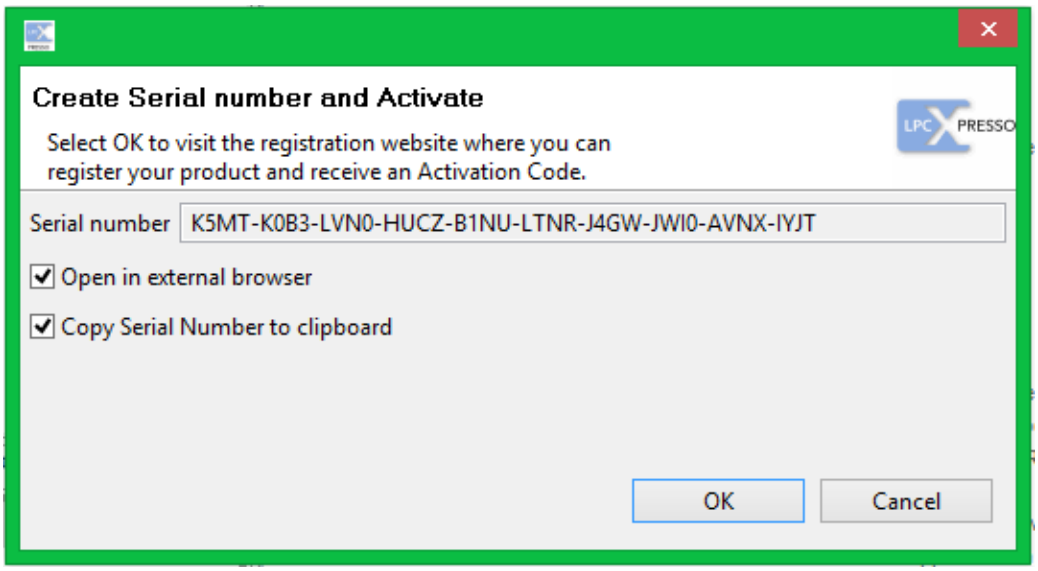
12. When the installation has finished. Select *Launch LPCXpresso* and click **Finish**.



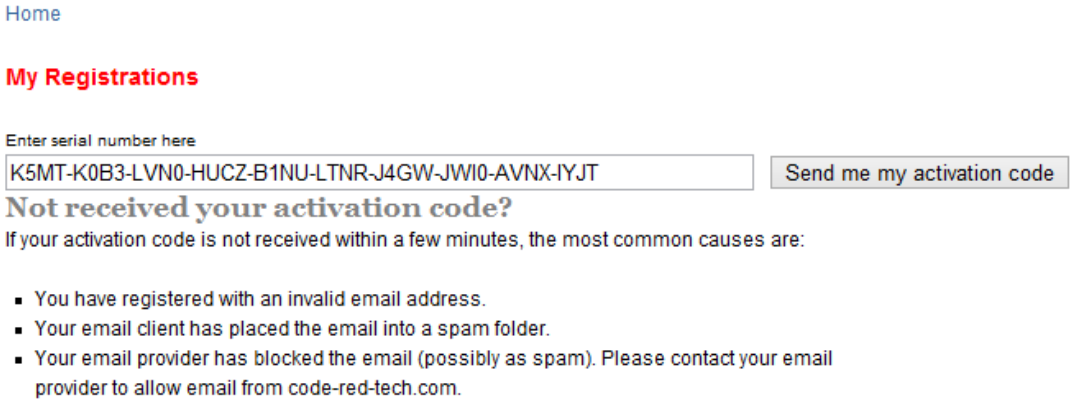
13. Once LPCXpresso launches, you will need to activate your copy by selecting **Help**-> **Product activation** -> **Create serial number and register**.



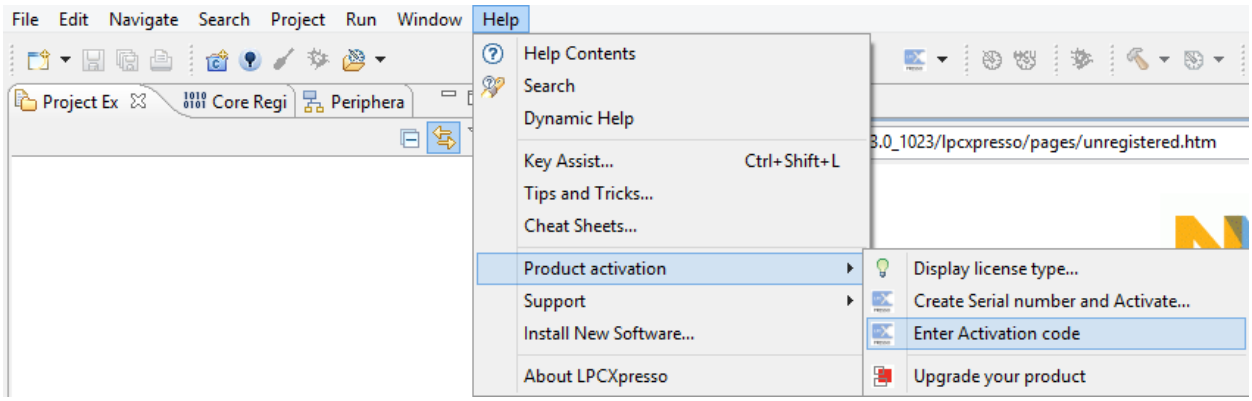
14. A dialog box will open with your serial number. Please select *Open in external browser* and *Copy Serial Number to clipboard* and then click **OK**.



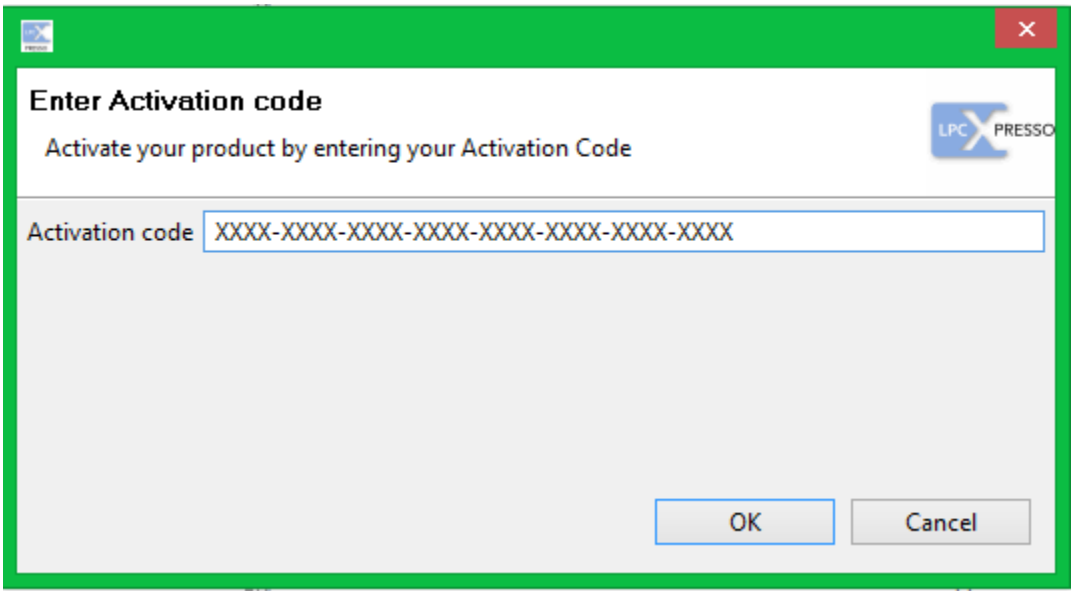
15. In your browser, click **Send me my activation code**. Then wait for a code to be sent to the email address provided for your account.



16. Once you receive your code via email, in LPCXpresso select **Help** -> **Product Activation** -> **Enter Activation Code**.



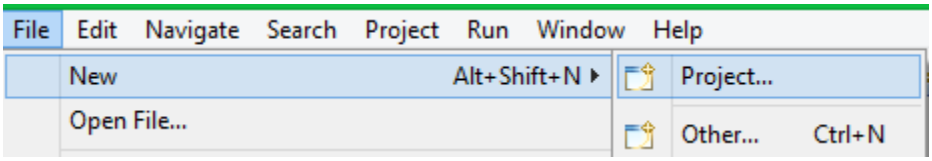
17. A dialog box will appear prompting you for your 32-digit activation code. Enter your code and click **OK**.



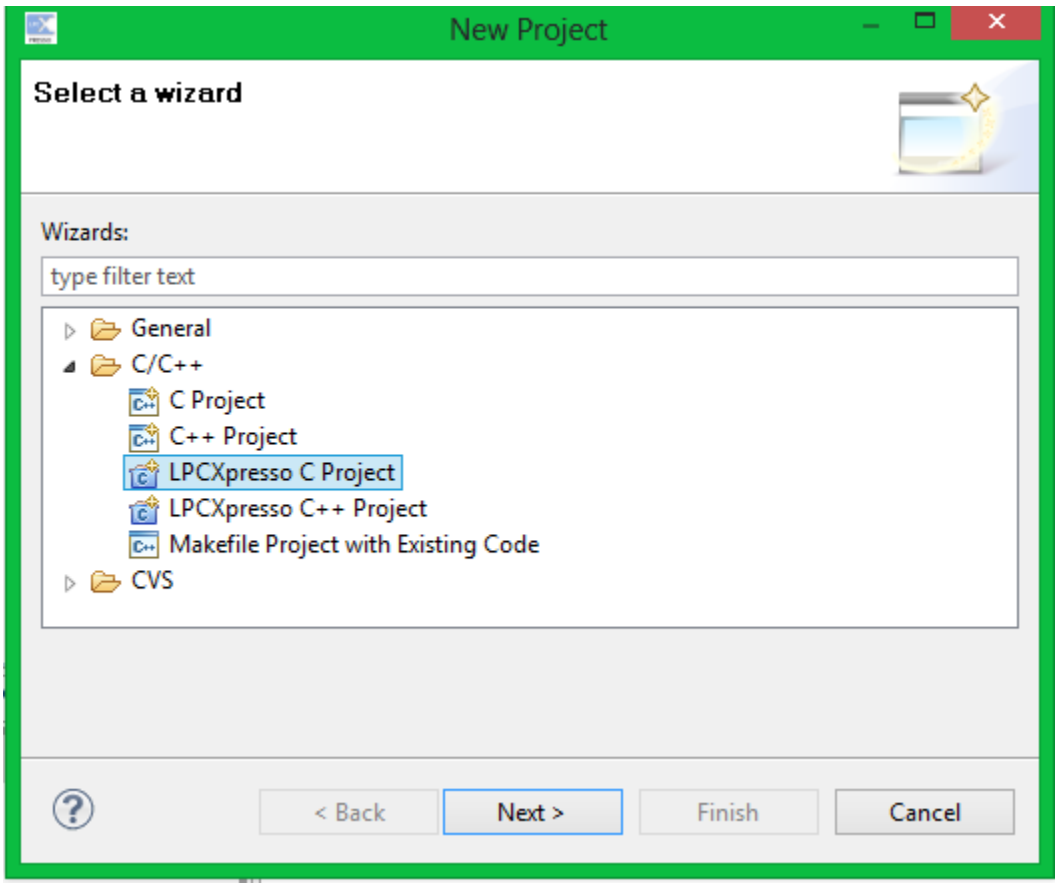
18. Congratulations! You have now successfully installed and registered your own copy of LPCXpresso.

E2.2 Creating a New Project

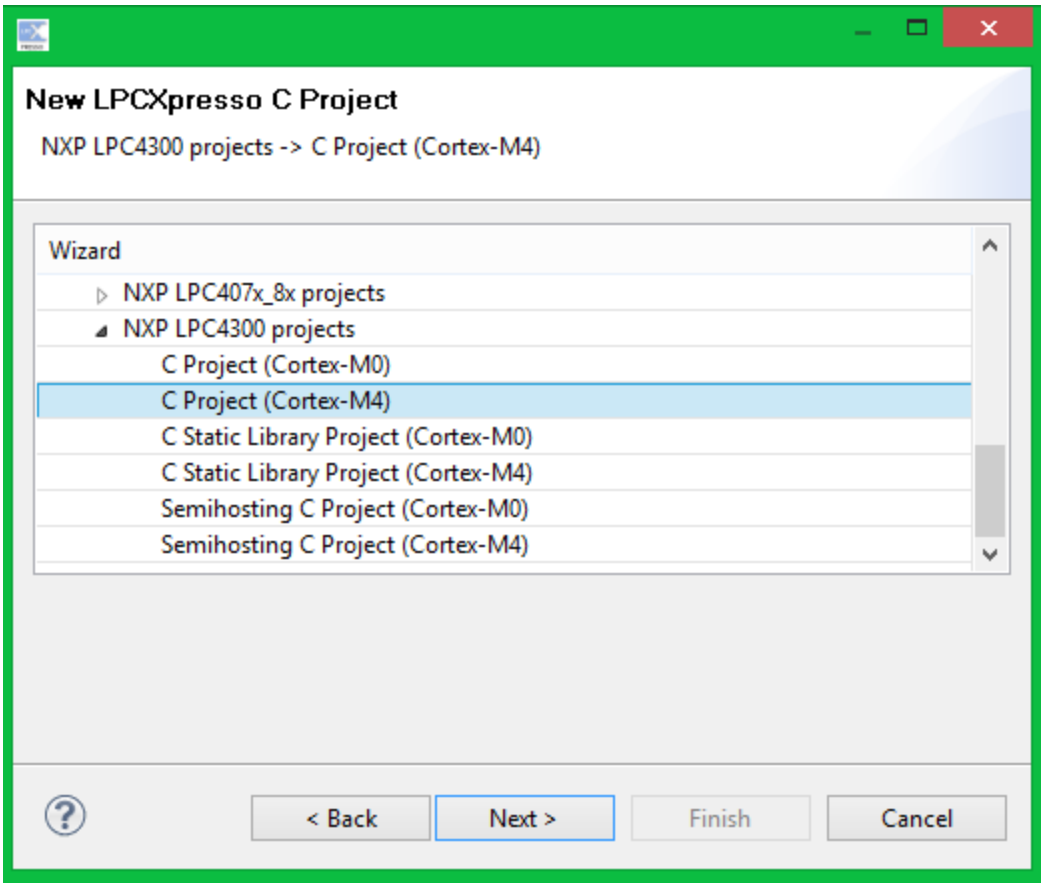
1. In order to create a new project in LPCXpresso, first select **File -> New -> Project**.



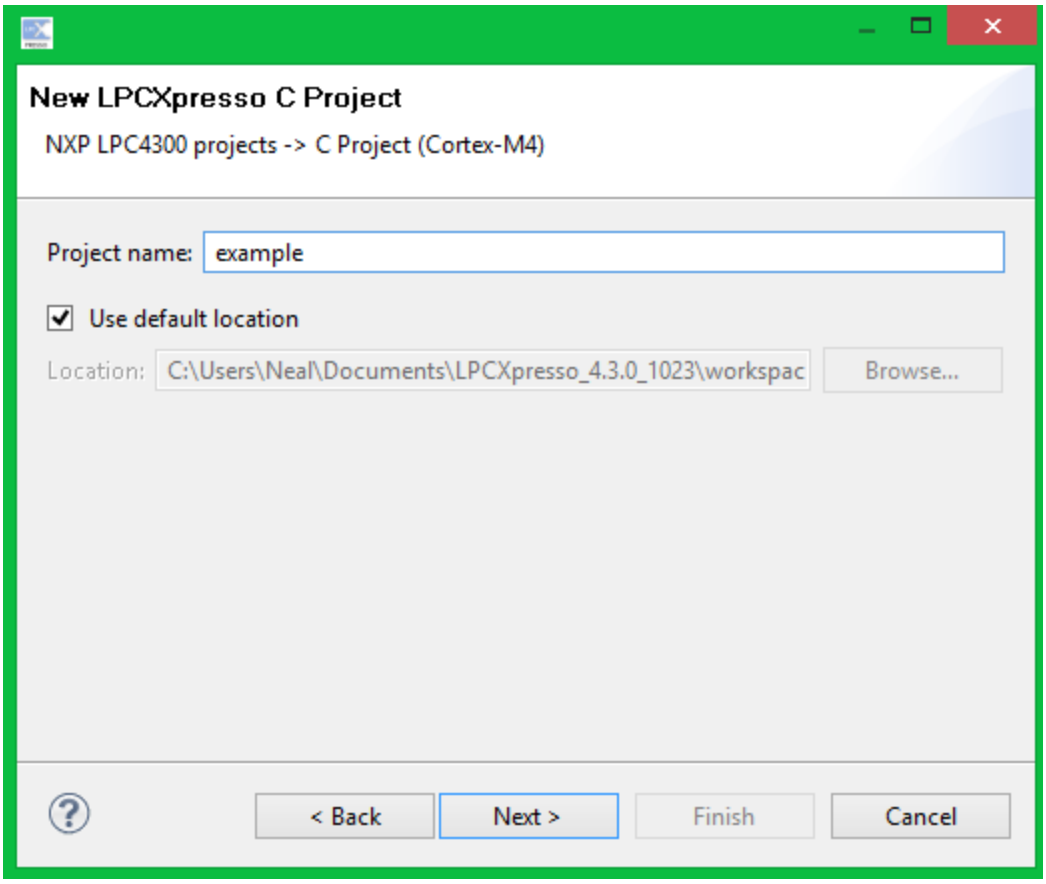
2. The New Project Wizard will then launch. Please Select **C/C++ -> LPCXpresso C Project** to begin creating your project. Then click **Next**.



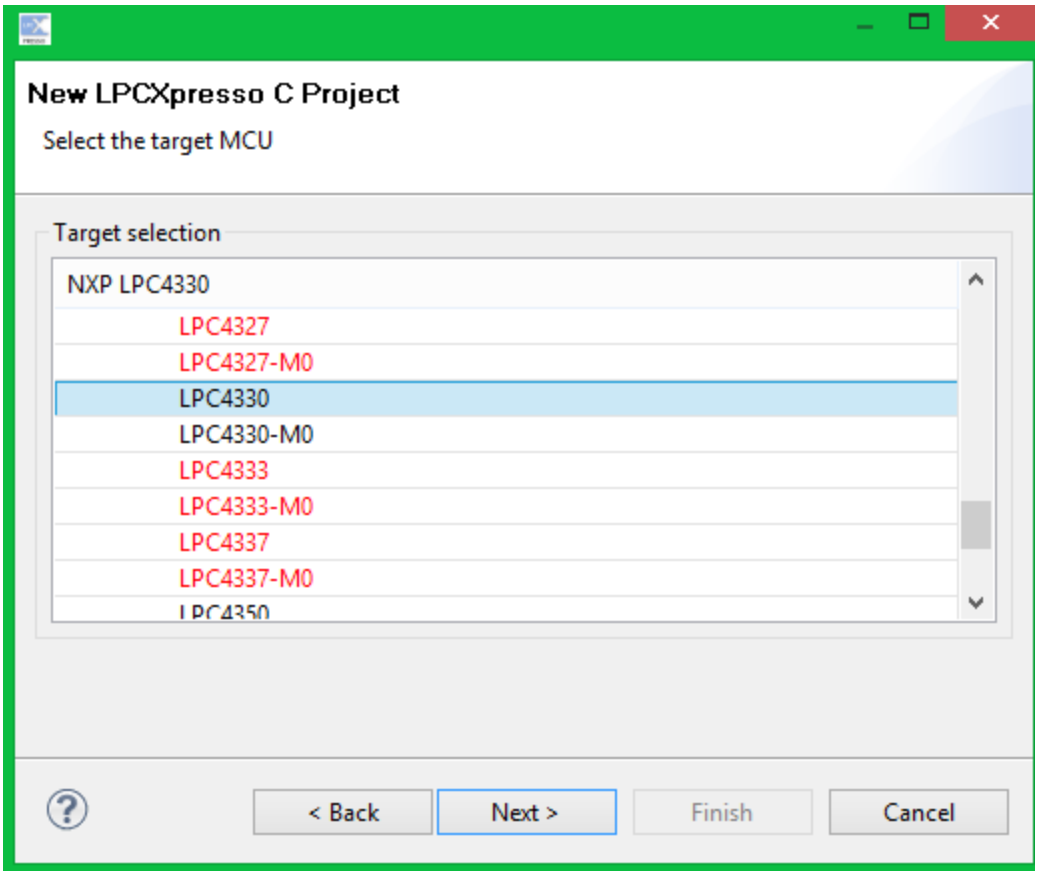
3. The Abstract Hardware Device utilizes the Cortex-M4 Core. Therefore, select **NXP LPC4300 projects -> C Project (Cortex-M4)**. Then click **Next**.



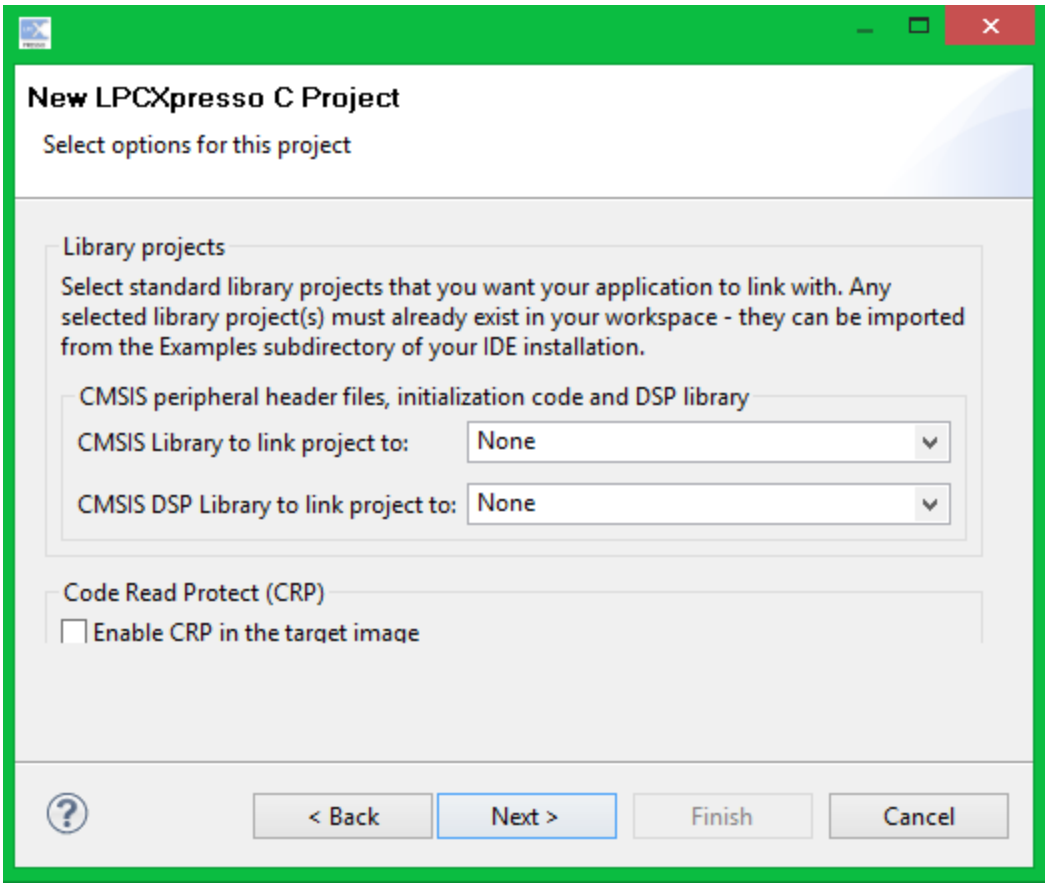
4. You will then be asked to specify a project name. By choosing to use the default location, the new project will be saved in your workspace. Name your project and click **Next**.



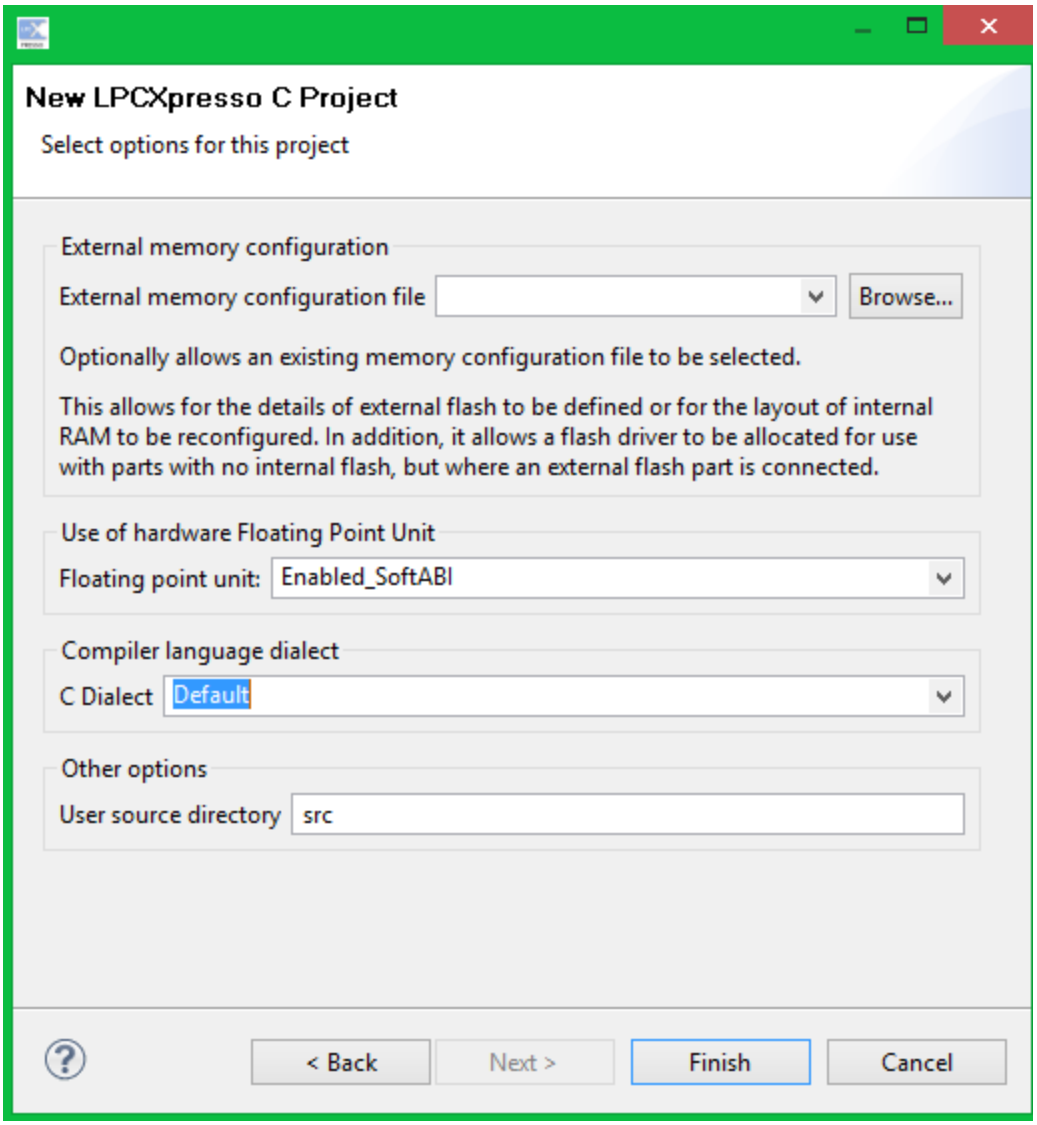
5. You will then be prompted to select an MCU to configure the project to. Select **LPC4330** as the target and click continue.



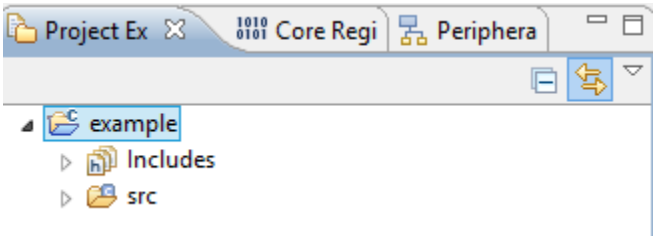
6. After selecting a target MCU, the wizard will ask for any CMSIS Library's that you would like to link the project to. Please select **None** for both drop-down menus. Also **deselect** the check-box to dis able CRP in the target image. Then click **Next**.



7. One last step before your new project is created. Do not select an external memory configuration file. Also select **Enabled_SoftABI** for the floating point unit, **Default** for C Dialect, and **src** for User source directory. Click **Finish**.

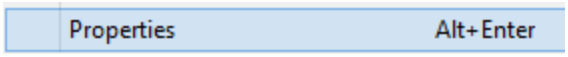


8. You have now successfully created a new LPCXplorer project. You can see the project folder and add new source files in the project explorer.

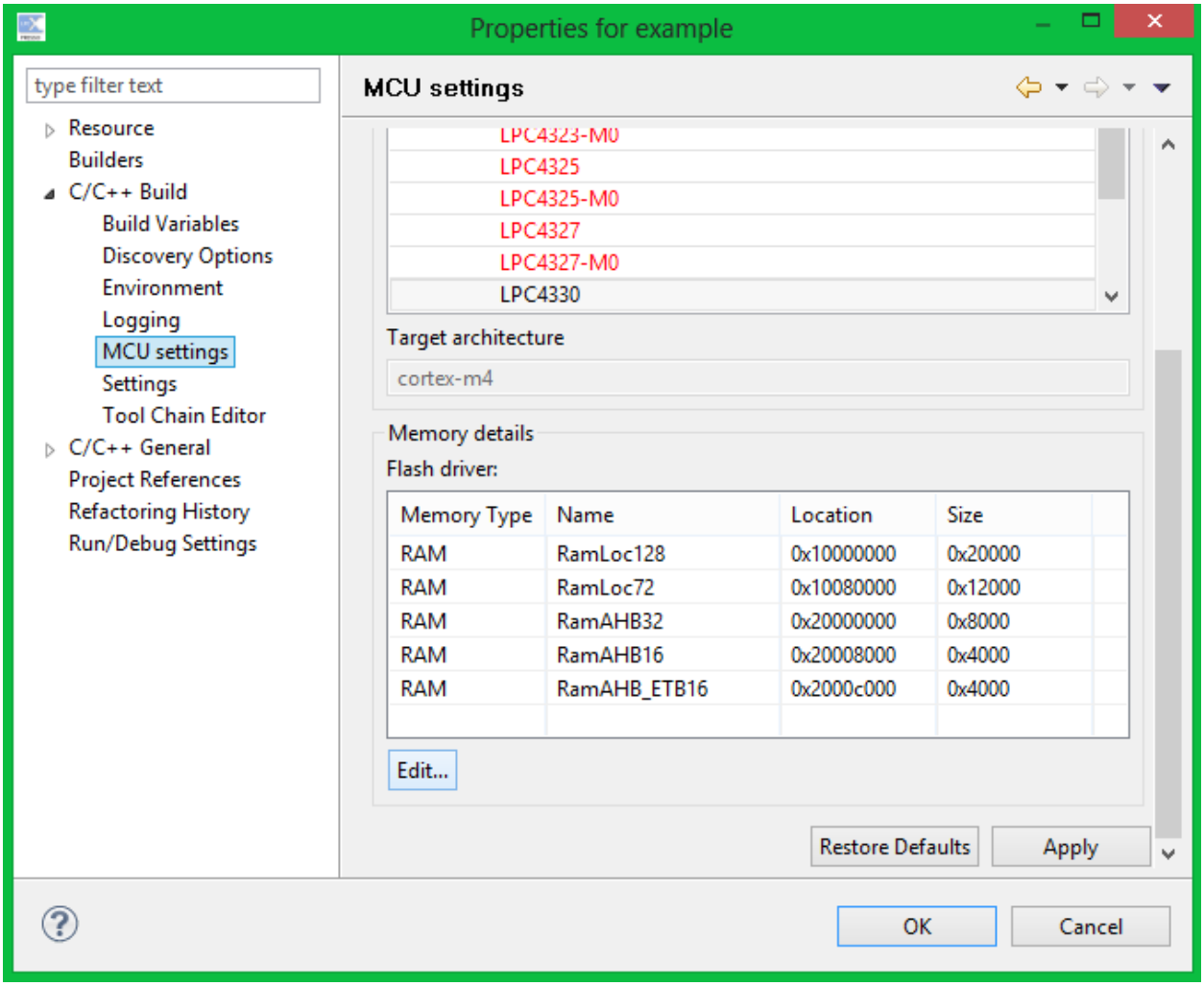


E2.3 Downloading Code via JTAG

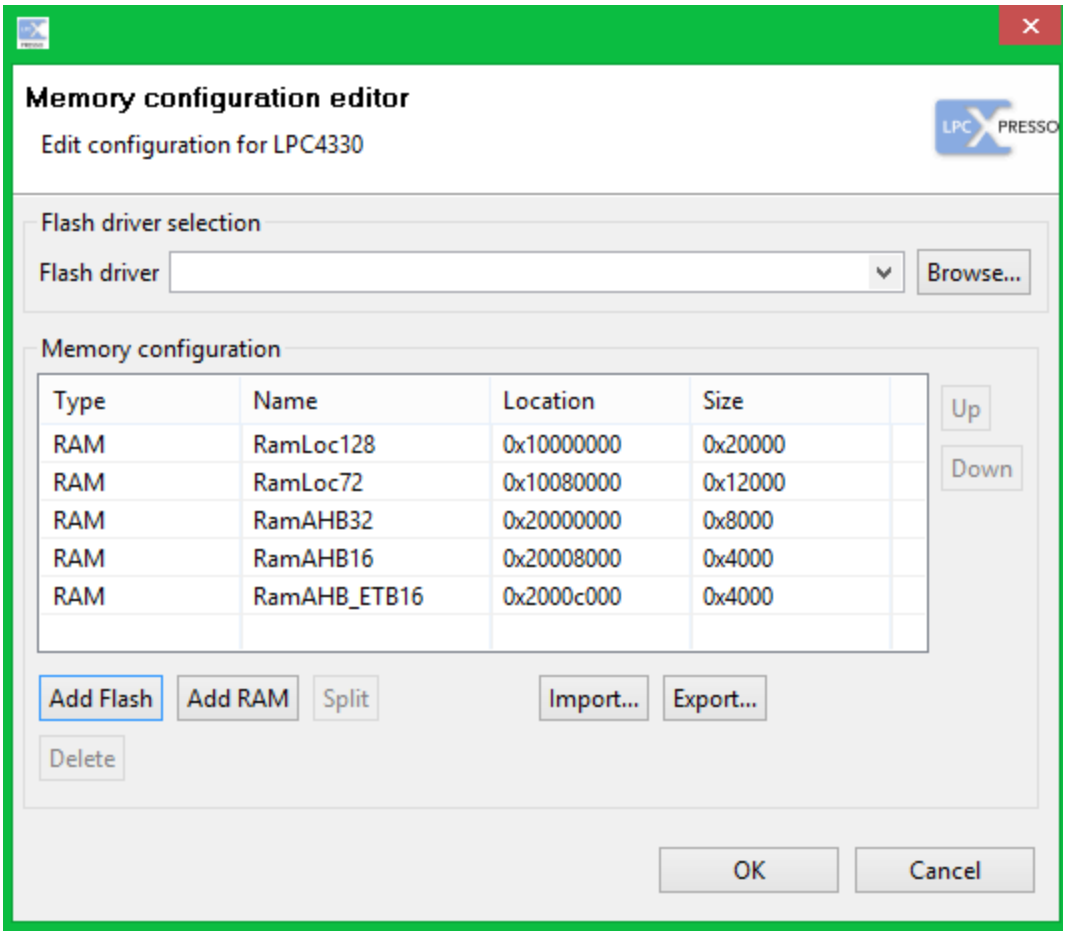
1. Before you can download software to the MCU, you must add external flash space to your project. Right click on your project and select **Properties**.



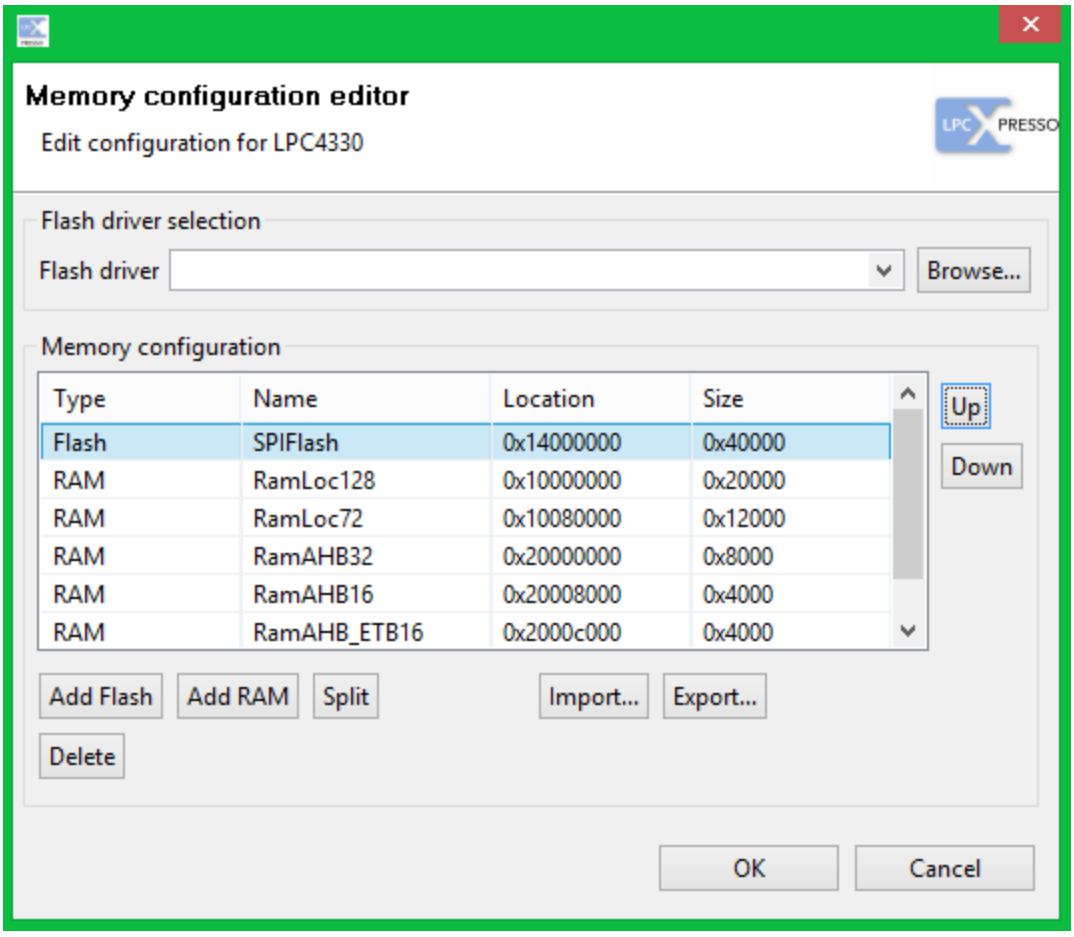
2. Expand **C/C++ Build** and select **MCU settings**. Under MCU settings notice the Memory Details for the flash driver. Click **Edit…**



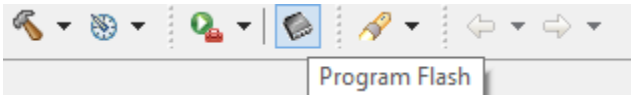
3. A new window should appear. Click **Add Flash**.



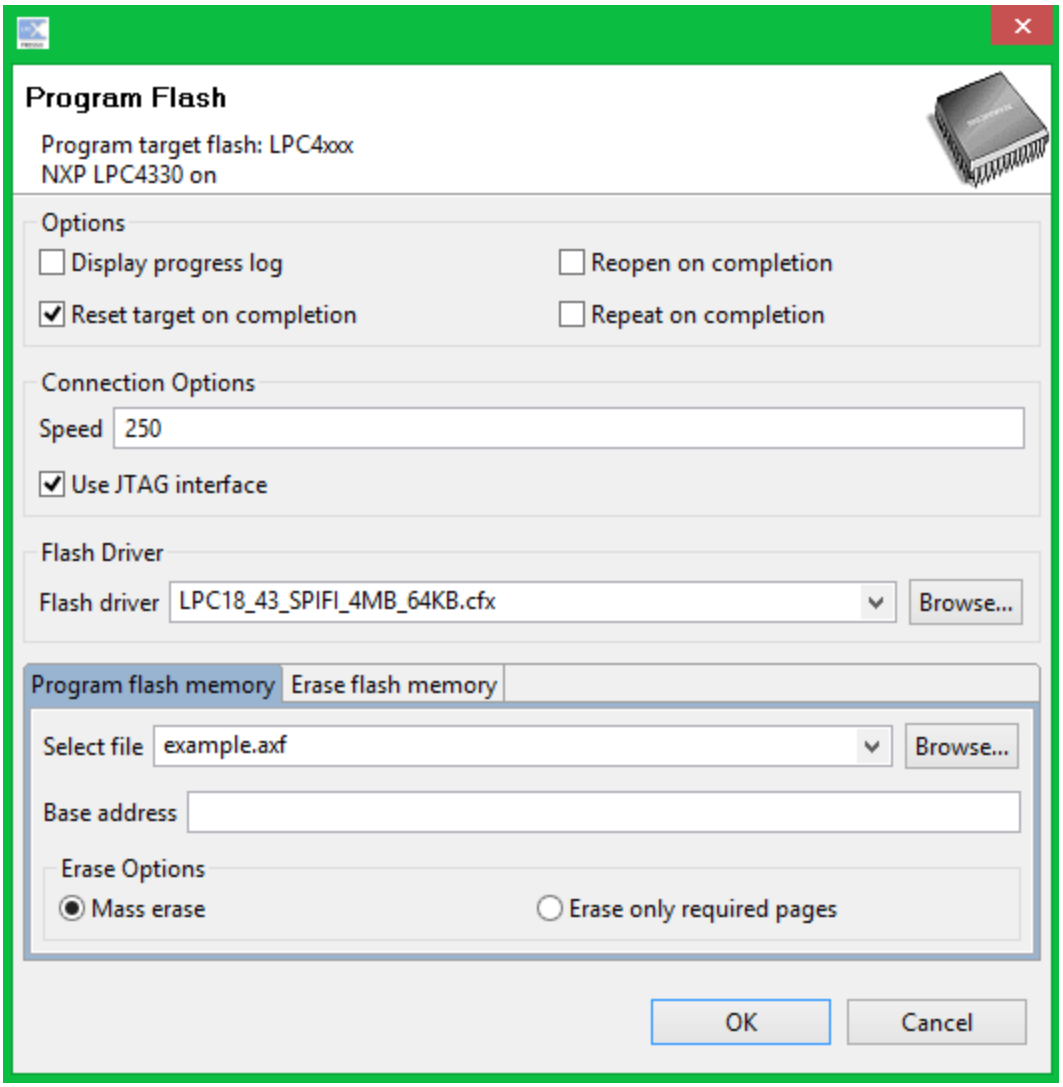
4. Notice a new line of Type Flash has been added to the bottom of the Memory Configuration table. Change the Name to **SPIFlash**, Location to **0x14000000**, and Size to **0x40000**. Then click **Up** five times until the new flash is at the top of the Memory Configuration. Click **OK** in both dialog windows.



5. Once you have your AHD and JTAG connected to your CPU, you are ready to begin downloading your project. To begin the process, click **Program Flash** icon from the top toolbar.



6. The Program Flash window will open. Select the **Use JTAG interface** checkbox, **LPC18_43_SPIFI_4MB_64KB.cfx** for the Flash Driver, **Mass erase** for erase options, and the **.axf file** generated by your project for the select file. Click **OK.**



7. You have now successfully downloaded code to your board and are able to explore the functionality of the AHD and LPCXpresso.

E3. Eclipse Setup

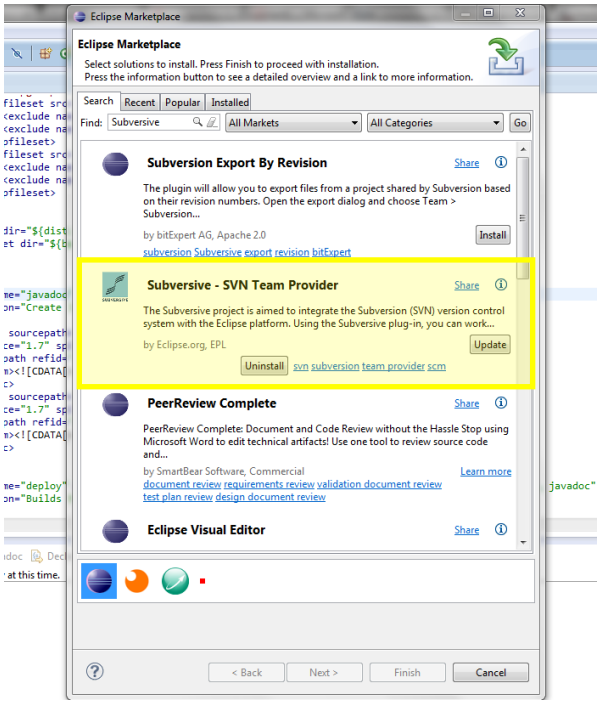
E3.1 Things you'll need:

1. The latest copy of eclipse, downloaded and extracted
    a. *http://www.eclipse.org/downloads/packages/*
    b. Download "Eclipse IDE for Java Developers"
    c. Extract wherever you like
2. An installed JDK (Java Development Kit) (1.6 or 1.7 should do fine)
3. A Google account
4. Internet Access

E3.2 Steps:

E3.2.1 Install Subversion

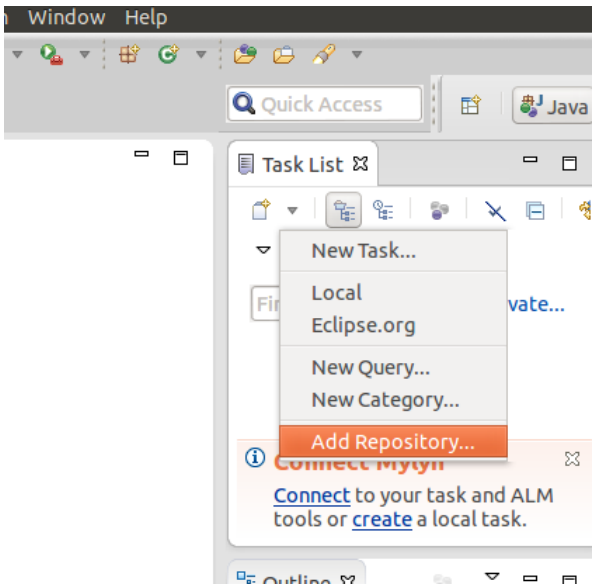
1. Open up eclipse.
2. Select your workspace location (DO NOT PUT WORKSPACE IN DROPBOX)
3. Help > Eclipse Marketplace > Search "Subversive"
4. Install the highlighted plug-in



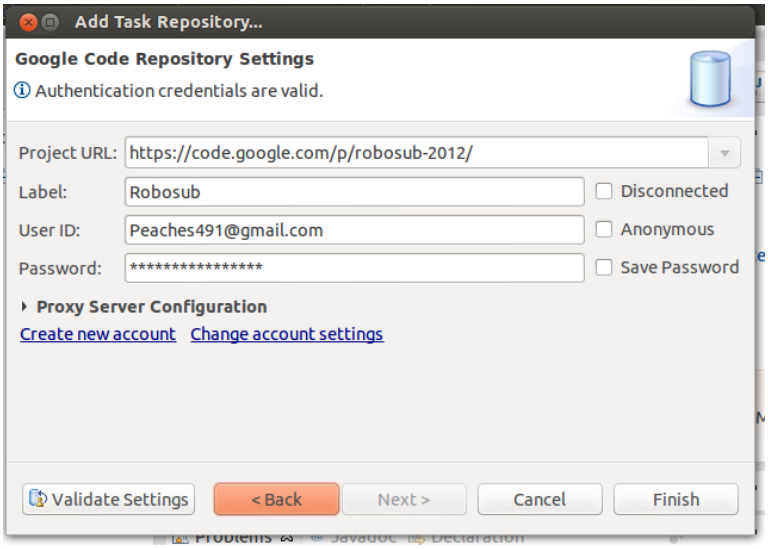
5. Restart Eclipse
6. When prompted To “Install Subversive Connectors” select all and install all

E3.2.2 Installing the Google Code Mylyn Connector (Task List and Bug tracking)

1. From Eclipse: Help > Install New Software
2. Enter *http://knittig.de/googlecode-mylyn-connector/update/*
    a. Instructions: *http://code.google.com/p/googlecode-mylyn-connector/*
3. Follow the instructions to install
4. Restart eclipse
5. On the task list drop down on the right, choose “Add Repository”



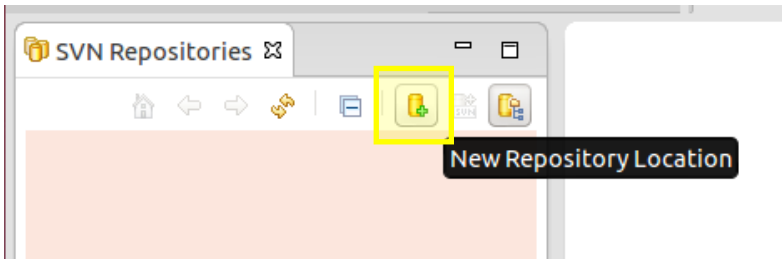
6. Select Google Code
7. Enter the following credentials (substituting your username and password)



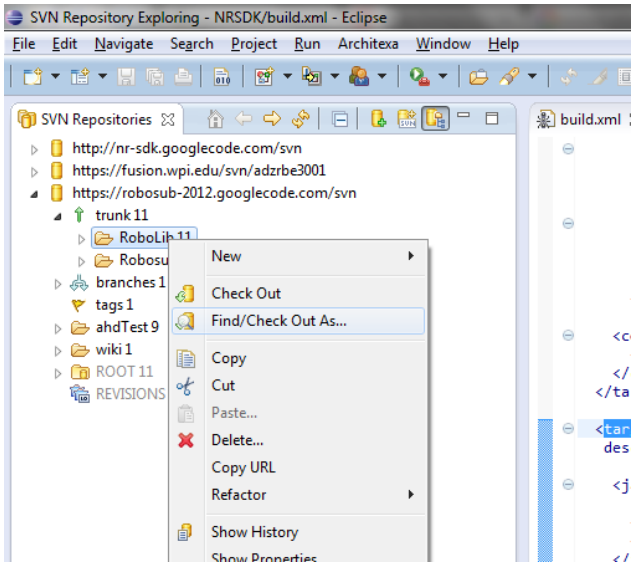
8. Yes, add a Query
9. Choose “Open Issues”

E3.2.3 Checking out the Repository

1. Navigate to *https://code.google.com/p/robosub-2012/source/checkout*
2. In Eclipse: Window > Open Perspective > Other > SVN Repository Exploring



3. Use the credentials specified by the Google Code page
    a. Use only the URL portion of the Write access section
    b. *https://robosub-2012.googlecode.com/svn/*
    c. Use the Google Code Generated password
4. Check out the project as…



5. Check out as a new project in your workspace

Appendix F: Various ME equations

F1. Hydrostatics and Hydrodynamics

The buoyant force is given by $F_B = \rho g V$, where F_B is the buoyant force, ρ is the density, g is the gravity, and V is the volume.

The simplified continuity equation is $V_1 A_1 = V_2 A_2$, where V is the velocity and A is the area for the entry and exit of a control volume.

For an incompressible flow, Bernoulli's Equation is $\frac{p}{\rho} + \frac{V^2}{2} + gz = constant$, where p is the pressure, ρ is the density of the fluid, V is the velocity, g is the acceleration of gravity, and z is the elevation relative to a reference plane.

The drag force is given by $F_D = \frac{1}{2}\rho C_D A_C V^2$, where F_D is the drag force, ρ is the density, C_D is the drag coefficient, A_C is the cross-sectional area perpendicular to the motion, and V is the velocity.

F2. Propulsion

The thrust is given by $F_{thrust} = \dot{m}(u_e - u)$, where F_{thrust} is the thrust, $\dot{m}$ is the change of mass with respect to time, and $(u_e - u)$ is the speed of the exhaust fluid relative to the moving object.

The power required to overcome drag is $= F_D V = \frac{1}{2}\rho C_D A_s V^3$.

F3. Heat Transfer

Heat transfer due to conduction is given by: $Q_{cond} = kA(T_h - T_c)$, where k is the thermal conductivity of the material, A is the area, T_h is the higher temperature, and T_c is the colder temperature.

Heat transfer due to convection is given by $Q_{conv} = hA(T_s - T_\infty)$, where h is the heat transfer coefficient, A is the area, T_s is the temperature of the surface of the object, and T_∞ is the temperature of the environment suitably far from the surface so as to not be affected by it.

Heat transfer from an object due to radiation is given by $Q_{rad} = \sigma \epsilon A T^4$, where σ is the Stefan-Boltzmann constant, ϵ is the emissivity of the object, A is the surface area of the emitting body, and T is the absolute temperature.

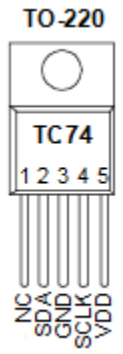
Appendix G: Ballast placement

The passive ballast needs to be added incrementally with equal portions attached to the frame as close to each of the 4 corners of the frame of WAVE as possible. The buoyant foam should be cut into cylinders of with a diameter between 5 cm to 10 cm depending on needs of the platform for the research being conducted. These cylinders should be distributed evenly around the top level of the frame, focusing on keeping distribution mirrored and balanced.

Appendix H: Sensors

For the sensors it is necessary to understand each sensor and how it will interface with the board design.

Temperature Sensor – Microchip TC 74

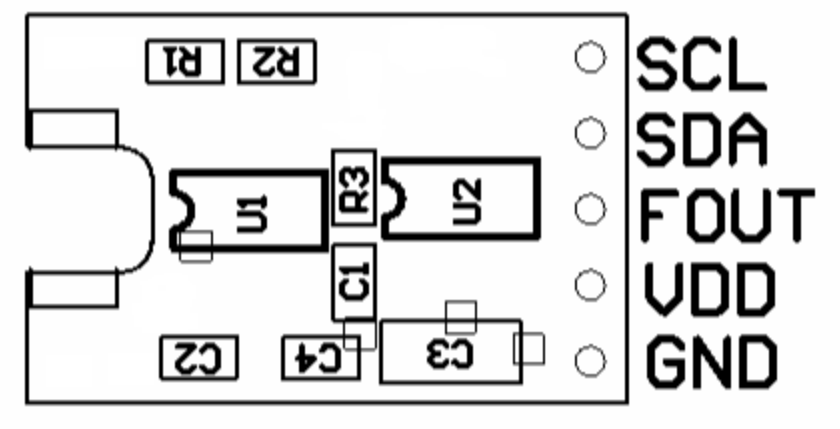


Reference Data Sheet for more Specification

Each temperature sensor communicates with a board via an I2C interface. In order to be able to use this you will need to set up the code necessary in order to use and I2C with the Arduino Leonardo Interface. In order to communicate via I2C you will have to follow the following instructions.

i. Plug in and Power up the sensor
ii. Look up the address of the sensor
iii. Use the Wire Library to set up the functionalities in order to communicate with the sensor.

Humidity Sensor- Sparkfun HD100



Reference Datasheet for more specifications

The humidity sensor also communicates with the I2C interface.

1) Plug in and Power up the sensor
2) Set Up the pins to communicate with a digital Interface such as the Arduino Leonardo
3) Use the Wire Library to set up the functionalities in order to communicate with the sensor.
4) Set up the Fout pin to communicate with PWM pin.
5) Use the following equation in you code in order to calculate the relative humidity.

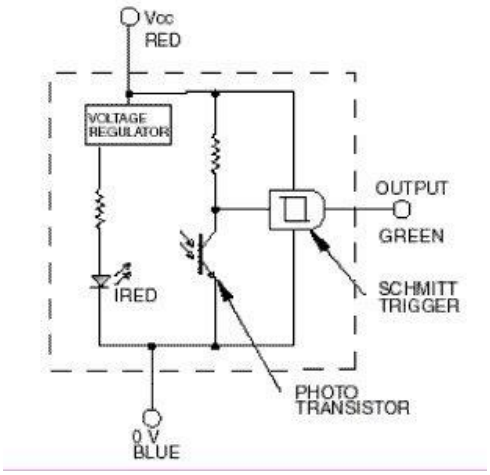
HH10D Humidity Calculation Algorithm

Data Definition		eeprom address
sensitivity	Sens(2byte value)	10
Offset	2 byte value	12
RH(%)=	(offset-Soh)*sens/2^12	

Pressure Sensor

In order to use the pressure sensor you must first make sure that the sensor is calibrated to the range in which you plan to measure upon. The pressure sensor that was used within the design was a loan from NEST and was calibrated to measure between 0 – 15 PSI.

Flood Sensor



In order to user the Honeywell Flood Sensor you must set up your microcontroller to be able to read digital outputs.

1) Plug in and Power up the sensor
2) Set Up the pins to communicate with a digital Interface such as the Arduino Leonardo
3) Set one pin to power, one to ground according to the pinout
4) Write code in order to retrieve values from the flood sensor.

Appendix I: Accessing Board Schematics

I.1: Design Files and PDF Of Abstract Hardware Device Schematic and Printed Circuit Board Layout

https://github.com/adamjvr/WAVE_Abstract_Hardware_Device

I.2: Design Files and PDF Of Thruster Controller Schematic and Printed Circuit Board Layout

https://github.com/adamjvr/WPI_WAVE_Thruster_Controller

Design Files and PDF Of AHD Shield Development Kit Schematic and Printed Circuit Board Layout

https://github.com/adamjvr/WPI_WAVE_AHDShieldHDK

Design Files and PDF Of Active Ballast Controller Schematic and Printed Circuit Board Layout

https://github.com/adamjvr/WAVE_Ballast_AuxAct_Controller

Design Files and PDF Of Power Supply Boards Schematic and Printed Circuit Board Layout

https://github.com/adamjvr/WPI_WAVE_PowerElectronics

Design Files and PDF Of Navigation & Locomotion Shield Schematic and Printed Circuit Board Layout

https://github.com/adamjvr/WAVE_Navigation_and_Locomotion_Shield

Altium File Viewer Download for viewing Altium files directly

http://www.altium.com/en/products/downloads

Appendix J: Code

Elements of both the embedded and java code have been discussed throughout the report, particularly in chapters 5, 6, and 8. Additionally, more details about the code can be found in: E2. LPCXpresso Manuals, E3. Eclipse Setup, Appendix N: RPCs, Appendix O: Example Mission Files, and Appendix P: Ant Build File .

In order to access the embedded level software, connect to the Neuron Robotics Bowler Communication Server SVN at http://nr-sdk.googlecode.com/svn/trunk/. Once access to the SVN has been established, select **trunk -> firmware -> device**. Inside this folder is all of the driver software. Test code such as the echo server can be accessed through the zip files.

To access the Java code you will first have to install a software plugin for eclipse by following the instructions at the following link: http://code.google.com/p/googlecode-mylyn-connector/. Then go to the task list and add a repository and select Google code. The credentials you need to access the database are:

- Project URL: https://code.google.com/p/robosub-2012/
- Label: Robosub
- Then use your personal Google account username and password

Appendix K: Bill of Materials

- Propellers
- PVC Connectors
- 80/20
- Prototype Parts
- Electronics Housing Tube and Plate
- (2) Ballast Pumps
- (2) Motor Shrouds
- (1) Rubber Diving Brick
- Sylgard-184 Silicone Elastomer, 2-part curing
- (1) Solid State Drive
- Flood Sensor
- (6) Motors
- Heat sinks
- Ethernet Pair Connector
- Waterproof Connectors
- (1) Schottky Diode
- (1) Fire Extinguisher
- Battery Connectors
- Battery Charger
- Thermal Paste
- Pololu DC-DC Converters
- Vicor DC-DC Converters
- (3) Gens Ace LiPo Batteries
- (1) Fit-PC

Appendix L: IP Rating

IP is an acronym for "Ingress Protection" against objects and water that intrude into the enclosure of any type of equipment. It is typically followed by two digits: the first gives an indication as to protection from solid objects and the second as to water, as can be seen from the diagram below. Since WAVE is a fully submergible craft, some of its components must have IP68 ratings in order to not take on water.

The IP ratings are defined in the IEC standard 60529, which was developed by the International Electro-technical Commission

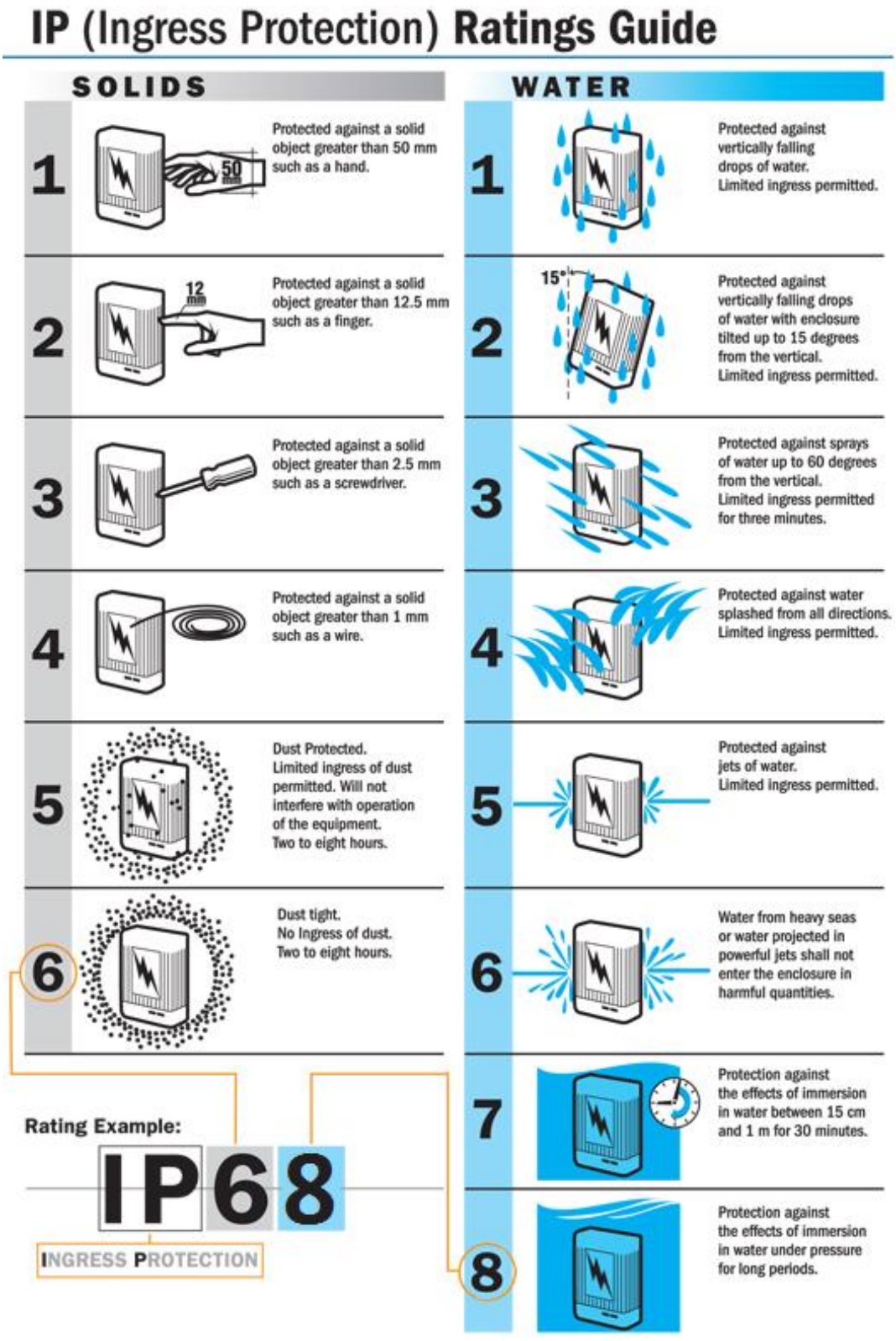


Figure 94: BlueSea IP Ratings [57]

Appendix M: Budget

M1. Department Breakdown

Table 25: Department Breakdown of Budget

Date	Department	Vendor	Cost	Balance
10/9/2012	ECE	Micro Controller Pros	63.71	2,016.29
11/1/2012	ECE	Sparkfun	15.44	2,000.85
11/7/2012	ECE	Digikey	33.00	1,967.85
11/7/2012	ECE	Sparkfun	43.59	1,924.26
11/19/2012	CS	Compulab	423.00	1,501.26
12/5/2012	ECE	Advanced Circuits	169.92	1,331.34
12/5/2012	ECE	Newark	39.37	1,291.97
12/5/2012	ECE	Digikey	9.54	1,282.43
12/5/2012	ECE	Mouser	192.69	1,089.74
12/10/2012	ECE	Infotech	25.42	1,064.32
12/12/2012	ECE	Ebay	11.98	1,052.34
12/12/2012	ECE	Digikey	44.63	1,007.71
1/15/2013	ECE	Vicor	22.35	985.36
1/24/2013	ECE	Acepower Electronics	67.60	917.76
2/1/2013	ECE	Mouser	255.44	662.32
2/5/2013	ECE	Ebay	84.88	577.44
2/7/2013	ME	American Micro Industries	124.59	452.85
2/12/2013	ECE	HobbyKing	42.66	410.19
2/12/2013	ECE	HobbyPartz	102.25	307.94
2/22/2013	ECE	Newark	26.32	281.62
2/22/2013	ECE	Mouser	476.72	-195.10
2/22/2013	ECE	Ebay	66.13	-261.23
2/22/2013	ECE	Digikey	11.97	-273.20
2/27/2013	ECE	Digikey	17.85	-291.05
2/27/2013	ECE	Pololu	95.37	-386.42

M2. Personal Contributions

Table 26: Personal Contributions

Vendor	Cost
HobbyPartz	$20.18
Home Depot	$8.00
Home Depot	$75.00
MetalsDepot	$182.97
Harbor Freight	$62.08
Home Depot	$8.09
Inavas Medical	$37.99
Lowes	$16.00
Amazon	$21.12
Amazon	$67.27
Mouser	$40.45
Amazon	$31.66
Ebay	$17.97
Ebay	$12.50
Ebay	$237.40
Semiconductor On	$11.00
Home Depot	$21.22
Ebay	$18.80
Turn 4 Hobbies	$41.35
Ebay	$7.79
HobbyPartz	$70.16
Pololu	$46.36
Staples	$12.00
Amazon	$158.30
80/20	$486.86

Appendix N: RPCs

N1. Battery

N1.1 Packet Format:

Request sent to the AHD will be of the following format:

```
[2012/1/30 21:57:55:993]  Debug : TX>>
       Raw Packet:     03 74 f7 26 00 00 00 10 00 05 a9 62 61 74 74
       Revision:       3
       Device ID:      74:F7:26:xx:xx:xx
       Packet Type:    GET
       Direction:      (0) Synchronous
       Reserved:       0
       Data Size:      4
       Checksum:       169
       RPC:            batt
       Data:           62 61 74 74
```

The response generated by the AHD will be of the following format:

```
[2012/1/30 21:57:55:993]  Debug : TX>>
       Raw Packet:     03 74 f7 26 00 00 00 10 00 05 a9 62 61 74 74 XX XX. . .
       Revision:       3
       Device ID:      74:F7:26:xx:xx:xx
       Packet Type:    POST
       Direction:      (0) Synchronous
       Reserved:       0
       Data Size:      n
       Checksum:       169
       RPC:            batt
       Data:           62 61 74 74 XX XX XX XX XX . . .
```

And the data (marked XX in the above diagram) will take the following format:

Byte	MSB							LSB
0	Number of Batteries							
1	Battery 1 Voltage (129 = 12.9V)							
2	Battery 1 Amperage (129 = 1.29A)							
3	Battery 1 Temperature (Degrees Fahrenheit)							
4	Battery 2 Voltage (129 = 12.9V)							
5	Battery 2 Amperage (129 = 1.29A)							
6	Battery 2 Temperature (Degrees Fahrenheit)							
7	Repeat for # batteries in Byte 0							

Total size of the data field will be:

$$4(opcode) + 1(batteries(n)) + 3(n) = 5 + 3n\ bytes$$

$$n := 3 \xrightarrow{yields} 14\ bytes$$

This data field will then be separated into appropriate fields in the Java program, and the data distributed appropriately. A value of -1 for byte 0 would indicate an error.

N2. Motor Velocity

N2.1 Packet Format:

Request sent to the AHD will be of the following format:

```
[2012/1/30 21:57:55:993]  Debug : TX>>
        Raw Packet:     03 74 f7 26 00 00 00 10 00 05 a9 6D 6F 74 72
        Revision:       3
        Device ID:      74:F7:26:xx:xx:xx
        Packet Type:    POST
        Direction:      (0) Synchronous
        Reserved:       0
        Data Size:      4
        Checksum:       169
        RPC:            motr
        Data:           6D 6F 74 72 XX XX XX XX. . .
```

The response generated by the AHD will be of the following format:

```
[2012/1/30 21:57:55:993]  Debug : TX>>
        Raw Packet:     03 74 f7 26 00 00 00 10 00 05 a9 6D 6F 74 72 XX
        Revision:       3
        Device ID:      74:F7:26:xx:xx:xx
        Packet Type:    STATUS
        Direction:      (0) Synchronous
        Reserved:       0
        Data Size:      n
        Checksum:       169
        RPC:            motr
        Data:           6D 6F 74 72 XX
```

And the data (marked XX in the above diagram) will take the following format:

Byte	MSB							LSB
0	Number of Motors							
1	Motor1 Duty Cycle and Direction							
2	Motor2 Duty Cycle and Direction							
3	Motor3 Duty Cycle and Direction							
4	Motor4 Duty Cycle and Direction							
5	Motor5 Duty Cycle and Direction							
6	Motor6 Duty Cycle and Direction							
7	Repeat for # motors in Byte 0							

Total size of the data field will be:

$$opcode + countByte + numMotors = datagramSize$$

$$4 + 1 + (n) = 5 + n$$

$$for\ n = 6 \rightarrow datagram = 11\ bytes$$

Directions for driving motors:

0 for a stopped motor

-127 for full reverse

128 for full forward

The additional byte specified in the response packet will indicate the number of motors set. This number should always equal the "n" form the first packet. If the original "POST" packet specifies more motors than the board can handle, it should return an error by replying with a -1 in this field.

N3. Emergency Stop

N3.1 Packet Format:

Request sent to the AHD will be of the following format:

```
[2012/1/30 21:57:55:993]  Debug : TX>>
        Raw Packet:     03 74 f7 26 00 00 00 10 00 05 a9 65 73 74 70 53 4F 53
        Revision:       3
        Device ID:      74:F7:26:xx:xx:xx
        Packet Type:    CRITICAL
        Direction:      (0) Synchronous
        Reserved:       0
        Data Size:      4
        Checksum:       169
        RPC:            estp
        Data:           65 73 74 70 53 4F 53
```

The packet will generate no response.

Byte	MSB							LSB
0	ASCII ‘S’							
1	ASCII ‘O’							
2	ASCII ‘S’							

This packet is used to indicate and emergency situation that requires the robot to surface and power down. It will be handled differently by each module.

N4. Twist

N4.1 Packet Format:

Request sent to the AHD will be of the following format:

```
[2012/1/30 21:57:55:993]  Debug : TX>>
        Raw Packet:     03 74 f7 26 00 00 00 10 00 05 a9 74 73 77 74
        Revision:       3
        Device ID:      74:F7:26:xx:xx:xx
        Packet Type:    POST
        Direction:      (0) Synchronous
        Reserved:       0
        Data Size:      4
        Checksum:       169
        RPC:            twst
        Data:           74 73 77 74
```

The response generated by the AHD will be of the following format:

```
[2012/1/30 21:57:55:993]  Debug : TX>>
        Raw Packet:     03 74 f7 26 00 00 00 10 00 05 a9 74 73 77 74 XX XX. . .
        Revision:       3
        Device ID:      74:F7:26:xx:xx:xx
        Packet Type:    STATUS
        Direction:      (0) Synchronous
        Reserved:       0
        Data Size:      n
        Checksum:       169
        RPC:            twst
        Data:           74 73 77 74 XX XX XX XX XX . . .
```

And the data (marked XX in the above diagram) will take the following format:

Byte	MSB							LSB
0-3	X Velocity (m/s)							

4-7	Y Velocity (m/s)
8-11	Z Velocity (m/s)
12-15	Angular Velocity about X-axis (rad/s)
16-19	Angular Velocity about Y-axis (rad/s)
20-23	Angular Velocity about Z-axis (rad/s)

Each four-byte parameter is an IEEE 754 32-bit floating point number. This fully defines the robot's velocity in six degrees-of-freedom.

Appendix O: Example Mission Files

The construction of mission files and task structures is discussed in detail in section 6.4.2. What follows is a selection of example mission files used during WAVE's testing.

```
<?xml version="1.0" encoding="UTF-8"?>
<Mission name="Test Mission 3">

    <Task type="WaitForGUI"/>

    <Task type="Echo" message="Waiting for 5 Seconds."/>
    <Task type="Wait">
        <Duration>5000</Duration>
    </Task>
    <Task type="Echo" message="Waiting for 5 Seconds."/>
    <Task type="Wait">
        <Duration>5000</Duration>
    </Task>
    <Task type="Echo" message="Waiting for 5 Seconds."/>
    <Task type="Wait">
        <Duration>5000</Duration>
    </Task>
    <Task type="Echo" message="Done!"/>
</Mission>
```

Figure 95: Echo Mission

The mission shown above simply tests if communication with the GUI is working by first waiting for a GUI connection. Once WAVE receives confirmation that a GUI has connected, it sends a series of 'echo' test messages, each separated by 5 seconds.

Appendix P: Ant Build File

```
<project default="run" name="WAVE RoboLib SDK" basedir="."
xmlns:svn="jwaresoftware.svn4ant.client">

	<property file="build.properties" />

	<property name="dir.src" value="src" />
	<property name="dir.build" value="build" />
	<property name="dir.data" value="data" />
	<property name="dir.doc" value="doc" />
	<property name="dir.media" value="media" />
	<property name="dir.dist" value="dist" />
	<property name="dir.test" value="test" />
	<property name="dir.lib-common" value="lib-common" />
	<property name="dir.lib-gui" value="lib-gui" />
	<property name="dir.lib-extra" value="${dir.lib-common}" />

	<!--Default properties for compiling the sub code-->
	<property name="main-class" value="com.robosub.main.Main" />
	<property name="project.name" value="RobosubLib-${app.version}" />
	<property name="jar.name" value="${project.name}.jar" />
	<property name="jarpath" value="${dir.dist}/${jar.name}" />
	<property name="dir.lib-extra" value="${dir.lib-common}" />

	<path id="build-classpath">
		<pathelement location="${jarpath}" />
		<fileset dir="${dir.src}" includes="**/*.jar" />
		<fileset dir="${dir.data}" includes="**/*" />
		<fileset dir="${dir.media}" includes="**/*" />
		<fileset dir="${dir.lib-common}" includes="**/*.jar" />
		<fileset dir="${dir.lib-gui}" includes="**/*.jar" />
		<fileset dir="utils/" includes="*.jar"/>

	</path>

	<path id="build-classpath-gui">
		<pathelement location="${jarpath-gui}" />
		<fileset dir="${dir.src}" includes="**/*.jar" />
		<fileset dir="${dir.data}" includes="**/*" />
		<fileset dir="${dir.media}" includes="**/*" />
		<fileset dir="${dir.lib-common}" includes="**/*.jar" />
		<fileset dir="${dir.lib-gui}" includes="**/*.jar" />
		<fileset dir="utils/" includes="*.jar"/>

	</path>


	<!--Code for setting references necessary to compile th GUI-->
	<property name="main-class-gui" value="com.robosub.gui.Main" />
	<property name="jar.name-gui" value="${project.name}-gui.jar" />
	<property name="jarpath-gui" value="${dir.dist}/${jar.name-gui}" />
```

```
	<!--################################## COMPILATION CODE
#####################################-->

	<target name="clean">
		<delete dir="${dir.build}" failonerror="false" />
		<delete dir="${dir.doc}" failonerror="false" />
		<delete dir="${dir.dist}" failonerror="true" />
	</target>

	<target name="prepare">
		<mkdir dir="${dir.build}" />
		<mkdir dir="${dir.dist}" />
	</target>


	<target name="compile" depends="prepare">

		<depend srcDir="${dir.build}" closure="true" />

		<!-- debuglevel="line" -->
		<javac srcdir="${dir.src}" destdir="${dir.build}" debug="on"
classpathref="build-classpath" includeantruntime="true" />

		<!-- <unzip dest="${dir.build}">
			<fileset dir="${dir.lib-common}">
				<include name="**/*.zip"/>
				<include name="**/*.jar"/>
			</fileset>
			<fileset dir="${dir.lib-extra}">
				<include name="**/*.zip"/>
				<include name="**/*.jar"/>
			</fileset>
		</unzip> -->

	</target>



	<!--################################## JAR CODE
#####################################-->

	<target name="jar">

		<manifestclasspath property="lib.list" jarfile="${jarpath}">
			<classpath refid="build-classpath" />
		</manifestclasspath>

		<jar destfile="${jarpath}" basedir="."
			excludes="**" filesetmanifest="mergewithoutmain">
			<zipgroupfileset dir="${dir.lib-common}" includes="**.jar" />

			<fileset dir="${dir.build}" includes="**/*" excludes="META-
INF/*.SF" />
			<fileset dir="." includes="${dir.media}/*" />
			<fileset dir="." includes="${dir.data}/*" />
```

```
                <exclude name="${dir.dist}/**"/>

                <manifest>
                        <attribute name="Main-Class" value="${main-class}" />
                        <attribute name="Class-Path" value="${lib.list}" />
                </manifest>
        </jar>

        <chmod file="${jarpath}" perm="+x" />
    </target>

    <target name="jar-gui">

        <manifestclasspath property="lib.list" jarfile="${jarpath-gui}">
                <classpath refid="build-classpath" />
        </manifestclasspath>

        <jar destfile="${jarpath-gui}" basedir="." excludes="**"
filesetmanifest="mergewithoutmain">
                <zipgroupfileset dir="${dir.lib-common}" includes="**.jar" />

                <fileset dir="${dir.build}" includes="**/*" excludes="META-
INF/*.SF" />
                <fileset dir="." includes="${dir.media}/*" />
                <fileset dir="." includes="${dir.data}/*" />
                <exclude name="${dir.dist}/**"/>

                <manifest>
                        <attribute name="Main-Class" value="${main-class-gui}" />
                        <attribute name="Class-Path" value="${lib.list}" />
                </manifest>
        </jar>

        <chmod file="${jarpath}" perm="+x" />
    </target>


    <!--################################### TEST AND RUN CODE
#####################################-->

    <target name="test" depends="run">
        <mkdir dir="${test.dir}" />
        <test destfile="${test.dir}/${ant.project.name}.test"
basedir="${build.dir}">
                <junit>
                        <classpath refid="classpath.test" />
                        <formatter type="brief" usefile="false" />
                        <test name="TestExample" />
                </junit>
        </test>
    </target>

    <target name="run">
        <echo message="Attempting to run!" />
        <java jar="${jarpath}" forked="true" args="" />
```

```
		<echo message="JAR launched." />
	</target>

	<target name="run-gui">
		<echo message="Attempting to run!" />
		<java jar="${jarpath-gui}"/>
		<echo message="JAR launched." />
	</target>


	<target name="compile-jar">
		<antcall target="compile"/>
		<antcall target="jar"/>
	</target>

	<target name="compile-jar-run">
		<antcall target="compile" />
		<antcall target="jar" />
		<antcall target="run" />
	</target>

	<target name="clean-compile-jar">
		<antcall target="clean"/>
		<antcall target="compile" />
		<antcall target="jar" />
	</target>

	<target name="clean-compile-jar-run">
		<antcall target="clean" />
		<antcall target="compile" />
		<antcall target="jar" />
		<antcall target="run" />
	</target>


	<!--################################### DOCUMENTATION AND DISTRIBUTION CODE
#######################################-->

	<!-- Generate javadocs for current project into ${dir.doc} -->
	<target name="javadoc" description="Generate program documentation">
		<mkdir dir="${dir.doc}" />
		<javadoc sourcepath="${dir.src}" destdir="${dir.doc}" />
	</target>

	<!-- Package javadocs into ${dir.dist} -->
	<target name="javadoc-jar" description="Generate program documentation">
		<jar basedir="${dir.doc}" destfile="${dir.dist}/${project.name}-
javadoc.jar" />
	</target>

	<!-- Package the source files into ${dir.dist} -->
	<target name="package-sources" description="Create source file JAR">
		<!-- <jar basedir="${dir.src}" destfile="${dir.dist}/${project.name}-
${project.version}-sources.jar" /> -->
```

```
            <jar basedir="${dir.src}" destfile="${dir.dist}/${project.name}-
sources.jar" />
      </target>

      <!-- Create all necessary Jar files and copy into ${dir.dist} -->
      <target name="do-release" depends="compile, jar, javadoc, javadoc-jar,
package-sources">
            <mkdir dir="${dir.dist}" />
            <copy file="LICENSE-2.0.html" todir="${dir.dist}" />
            <copy todir="${dir.dist}">
                  <fileset dir="${dir.doc}" includes="**" />
            </copy>
      </target>

      <target name="upload">
            <taskdef classname="net.bluecow.googlecode.ant.GoogleCodeUploadTask"
classpath="utils/ant-googlecode-0.0.3.jar" name="gcupload"/>
            <gcupload
                        verbose="true"
                  username="Peaches491@gmail.com"
                  password="fX2nX6mJ3mb8"
                  projectname="robosub-2012"
                        targetfilename="${jar.name}"
                     summary="Version ${app.version} of ${app.name}"
                  filename="${jarpath}"/>
      </target>
      <target name="upload-gui">
            <taskdef classname="net.bluecow.googlecode.ant.GoogleCodeUploadTask"
classpath="utils/ant-googlecode-0.0.3.jar" name="gcupload"/>
            <gcupload
                        verbose="true"
                  username="Peaches491@gmail.com"
                  password="fX2nX6mJ3mb8"
                  projectname="robosub-2012"
                        targetfilename="${jar.name-gui}"
                     summary="Version ${app.version} of ${app.name} GUI"
                  filename="${jarpath-gui}"/>
      </target>

      <target name="fitpc-upload" description="upload jar file to FitPC">
            <!-- Must add
                  utils/ant-commons-net.jar &
                  utils/commons-net-3.2.jar to ANT classpath -->
            <ftp
                  server="robosub.dyndns.org" port="12122"
                  userid="sub"
                  password="Robosub1!"
                  action="send" verbose="yes" >

                  <fileset file="${jarpath}" />
            </ftp>
      </target>
</project>
```